ADVANCED TIMING AND SYNCHRONIZATION METHODOLOGIES FOR DIGITAL VLSI INTEGRATED CIRCUITS

by

Baris Taskin

B.S. in Electrical and Electronics Engineering, Middle East Technical University, 2000

M.S. in Electrical Engineering, University of Pittsburgh, 2003

Submitted to the Graduate Faculty of
the School of Engineering in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2005

UNIVERSITY OF PITTSBURGH

SCHOOL OF ENGINEERING

This dissertation was presented

by

Baris Taskin

It was defended on

July 6th 2005

and approved by

Ivan S. Kourtev, Assistant Professor, Department of Electrical and Computer Engineering

Brady Hunsaker, Assistant Professor, Department of Industrial Engineering

Alex K. Jones, Assistant Professor, Department of Electrical and Computer Engineering

Steven P. Levitan, Professor, Department of Electrical and Computer Engineering

Marlin H. Mickle, Professor, Department of Electrical and Computer Engineering

Dissertation Director: Ivan S. Kourtev, Assistant Professor, Department of Electrical and Computer Engineering

ABSTRACT

ADVANCED TIMING AND SYNCHRONIZATION METHODOLOGIES FOR DIGITAL VLSI INTEGRATED CIRCUITS

Baris Taskin, PhD

University of Pittsburgh, 2005

This dissertation addresses timing and synchronization methodologies that are critical to the design, analysis and optimization of high-performance, integrated digital VLSI systems. As process sizes shrink and design complexities increase, achieving timing closure for digital VLSI circuits becomes a significant bottleneck in the integrated circuit design flow. Circuit designers are motivated to investigate and employ alternative methods to satisfy the timing and physical design performance targets. Such novel methods for the timing and synchronization of complex circuitry are developed in this dissertation and analyzed for performance and applicability.

Mainstream integrated circuit design flow is normally tuned for zero clock skew, edge-triggered circuit design. Non-zero clock skew or multi-phase clock synchronization is seldom used because the lack of design automation tools increases the length and cost of the design cycle. For similar reasons, level-sensitive registers have not become an industry standard despite their superior size, speed and power consumption characteristics compared to conventional edge-triggered flip-flops.

In this dissertation, novel design and analysis techniques that fully automate the design and analysis of non-zero clock skew circuits are presented. Clock skew scheduling of both edge-triggered and level-sensitive circuits are investigated in order to exploit maximum circuit performances. The effects of multi-phase clocking on non-zero clock skew, level-sensitive circuits are investigated leading to advanced synchronization methodologies. Improvements

in the scalability of the computational timing analysis process with clock skew scheduling are explored through partitioning and parallelization.

The integration of the proposed design and analysis methods to the physical design flow of integrated circuits synchronized with a next-generation clocking technology—resonant rotary clocking technology—is also presented. Based on the design and analysis methods presented in this dissertation, a computer-aided design tool for the design of rotary clock synchronized integrated circuits is developed.

Keywords: Clock Skew Scheduling, Level-Sensitive Circuits, Linear Programming, Resonant Clocking, Synchronization, Time Borrowing, Timing.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

The timing requirements of high-performance VLSI systems are becoming increasingly complex as both the size and complexity of these systems continue to grow. In order to satisfy these increasingly stringent timing and power requirements, conventional circuit design and electronic design automation techniques must be continuously revisited, modified and improved [41]. The timing of nano-scale circuits is getting more complex than ever, and novel clocking and synchronization technologies are emerging to replace conventional synchronization schemes. These clocking and synchronization technologies are often implemented in an ad hoc fashion because the essential automation infrastructure is non-existent. Alternative circuit design techniques, which are not integral to the mainstream design flow, are also used in an ad hoc fashion in order to meet specific design budgets. Electronic design automation is essential for the transition of these alternative design and synchronization techniques into the mainstream.

In this dissertation, advanced timing and synchronization methodologies that aim to satisfy the stringent timing budgets of high performance integrated circuits are presented. The presented methodologies make use of level-sensitive latches as storage elements in high performance integrated circuits as opposed to traditional, edge-triggered flip-flops. Clock skew scheduling and multi-phase clocking techniques are used in these level-sensitive circuits in order to further improve the circuit performance. These novel timing methodologies presented here are supported by fully-automated design and analysis techniques, satisfying the increasing need for electronic design automation.

To demonstrate, the physical design flow of rotary resonant clocking technology [96] is developed using the presented advanced timing and synchronization methodologies. This automated physical design flow enables the design and analysis of rotary-clock synchronized

integrated circuits in a reasonable amount of time. The performance of these circuits are superior compared to conventional circuits, due to both synchronization with rotary clocking technology and the integration of the presented timing and synchronization methodologies. The physical design flow for rotary clock synchronized circuits is implemented in a computer-aided design tool called "hpictiming".

1.1 MOTIVATION

The work presented in this dissertation is motivated by the increasing *design technology productivity gap* [35] between the CAD tools and the current manufacturing technology potential. This gap occurs because the limits of semiconductor manufacturing technology cannot be fully exploited by the current state-of-the-art design technology. The gap continues to widen because of the lack or inefficiency of available automated design support for design and analysis methodologies that permit higher circuit performances.

In this work, advanced timing and synchronization methodologies, promoting the utilization of level-sensitive latches and non-zero clock skew synchronization schemes, are presented. Level-sensitive latches permit higher operating frequencies, dissipate less power and are smaller in size compared to industry-standard edge-triggered flip-flops [43]. However, the design and analysis of level-sensitive circuits have been traditionally considered more complicated from a computational perspective.

Similarly, clock skew scheduling permits higher operating frequencies compared to industry-standard zero clock skew designs. Clock skew scheduling is not very popular in mainstream design flow mainly due to the complexity it introduces in the design of traditional clock tree networks [26]. Next-generation clocking technologies inherently permit (and even require) non-zero clock skew synchronization [10, 11, 62, 93, 94, 96, 97]. However, their implementation is hindered due to the scalability of the automation of clock skew scheduling.

The work presented in this dissertation is performed with one broad objective: *To demonstrate the electronic design automation of high-performance VLSI circuits implemented with the presented advanced timing and synchronization methodologies.*

1.2 PROBLEM STATEMENT

In this dissertation, advanced timing and synchronization methodologies that are critical to the design and analysis of high-performance VLSI circuits are presented. The presented advanced timing and synchronization methodologies not only improve the results achievable with conventional design methodologies but also constitute an integral part of the design flow for circuits synchronized with next-generation clocking technologies. The two main problems solved in this dissertation are:

1. **Developing automated design and timing analysis methods for non-zero clock skew, level-sensitive circuits which are synchronized by single and multi-phase synchronization schemes,**
2. **Integrating these methods into the physical design flow of circuits synchronized by the resonant rotary clocking technology.**

   In order to solve these problems, the following tasks are performed:

- *Develop an automated circuit design and analysis technique that targets non-zero clock skew, level-sensitive circuits.* Currently, there are no available automated design or analysis techniques that target non-zero clock skew, level-sensitive circuits.
- *Investigate and mitigate the limitations on the performance of conventional clock skew scheduling techniques.* An automated procedure to enable minor modifications to the logic network of a non-zero clock skew circuit (in order to improve the results of clock skew scheduling) is investigated.
- *Enhance the non-zero clock skew, level-sensitive design and analysis technique to accommodate for multi-phase synchronization schemes.* The effects of multi-phase synchronization on non-zero clock skew, level-sensitive circuits are investigated.
- *Analyze resonant clocking technologies to demonstrate their amenability to multi-phase level-sensitive circuits with non-zero clock skew.* The applicability of the advanced timing and synchronization methodologies described in this dissertation to next-generation clocking technologies is investigated.

- *Design and implement the physical design flow for circuits synchronized with resonant rotary clocking technology in a CAD tool.* The physical design flow to implement multi-phase, non-zero clock skew circuits that are synchronized by rotary clocking technology is described. Seamless integration of the advanced timing and synchronization methodologies into the physical design flow is pursued.

  In the dissertation, the details of each task are presented in dedicated chapters.

1.3 RESEARCH PLAN

The research presented in this dissertation is completed in two main phases. The first phase is the *theoretical development* phase. In this phase, the advanced timing and synchronization methodologies presented in this dissertation are developed. The methodologies are mathematically formulated for synchronous circuit models. Performance metrics are defined for each timing and synchronization methodology. The practicality and performance of each methodology are demonstrated on ISCAS'89 benchmark circuits with computer simulations.

The second phase is the *practical implementation and integration* phase. In this phase, the presented advanced timing and synchronization methodologies are integrated in the CAD tool "hpictiming". Hpictiming is developed to automate the physical design flow of circuits synchronized with the resonant rotary clocking technology. Experiments with hpictiming are performed on ISCAS'89 benchmark circuits and an industrial circuit to demonstrate the applicability of the advanced timing and synchronization methodologies on circuits synchronized with this next-generation clocking technology.

1.4 ORGANIZATION OF THE DISSERTATION

The rest of this dissertation is organized as follows. The operation and computer representation of a static CMOS-based synchronous digital integrated circuit system are presented in Chapter 2. In Chapter 3, the timing properties of a synchronous system and the ba-

sic principles of synchronous circuit operation are summarized. In Chapter 4, one of the pivotal advanced timing methodologies discussed in the dissertation, clock skew scheduling, is revisited. Also in Chapter 4, the application of clock skew scheduling on level-sensitive circuits is described with a linear programming approach to solve the popular clock period minimization problem. In Chapter 5, the delay insertion method proposed to improve the performance of non-zero clock skew systems is presented. In Chapter 6, the timing analysis framework presented in Chapter 4 is enhanced in order to accommodate for multi-phase synchronization schemes. In this context, the effects of multi-phase synchronization on non-zero clock skew, level-sensitive circuits are analyzed. In Chapter 7, the implementation and operation characteristics of a set of next-generation clocking technologies—resonant clocking technologies—are surveyed. A particular type of resonant clocking technology, rotary clocking technology, is described in detail. Rotary clocking technology is shown to inherently permit non-zero clock skew operation of circuits with fine grain skew control and highly-improved operation characteristics. In Chapter 8, the physical design flow of circuits synchronized with the rotary clocking technology is described. Also in Chapter 8, the development of a CAD tool for the implementation of this physical design flow is explained in detail. Conclusions are offered in Chapter 9. Finally, possible directions for future research are discussed in Chapter 10.

2.0 SYNCHRONOUS DIGITAL VLSI SYSTEMS

VLSI is an acronym that stands for very large scale integration. This term is used to refer to a broad area of electrical and computer engineering applications, where the focus is on the design and analysis of large-scale electronic integrated circuits on semiconductors [6, 12, 38, 59, 87, 90]. The design and analysis of systems such as high-performance computational elements, high-speed memory elements, sequential and combinational control logic units and sub-micron size analog circuits are exemplary to areas related to VLSI circuit design.

VLSI circuits can be classified according to their application areas, the manufacturing process technology, the operational characteristics and other features. In this dissertation, *digital VLSI circuit design* is of particular interest and in the rest of the document, the term *VLSI design* is used to refer to digital synchronous circuit design in nanometer scale unless otherwise indicated.

Synchronous digital VLSI circuits are composed of sequential and combinational circuit blocks. Sequential blocks consist of *logic* and *register* (or *storage*) elements while combinational blocks consist only of the former. A detailed discussion of sequential and combinational circuit blocks is presented in a variety of references such as [66]. All synchronous circuits have a well-defined ordering of *switching* events that ensures the correct ordering of data propagation among register elements and ultimately from inputs to outputs. Synchronous systems are very popular in system design due to their relative simplicity in the design and analysis stages. The sequentiality in time of data transfers in *synchronous* circuits are regulated by a globally distributed synchronization signal. The globally distributed synchronization signal is called the *clock* signal and defines the *timing scheme* or *timing discipline* of the synchronous circuit [26]. The clock signal is distributed throughout the circuit in order to generate and distribute the time reference to each register. The distribution of the clock

signal is accomplished through a highly specialized structure called the *clock distribution network* or *clock tree network* [26].

A digital synchronous circuit is a network of combinational logic elements and globally clocked registers. The combinational logic elements implement the functionality of the circuit. The clocked registers serve as storage elements to store the data computation results at each clock cycle. The logical order of the computation and storage processes is moderated by the globally distributed clock signal. An overall representation of a typical synchronous circuit is shown in Figure 1.

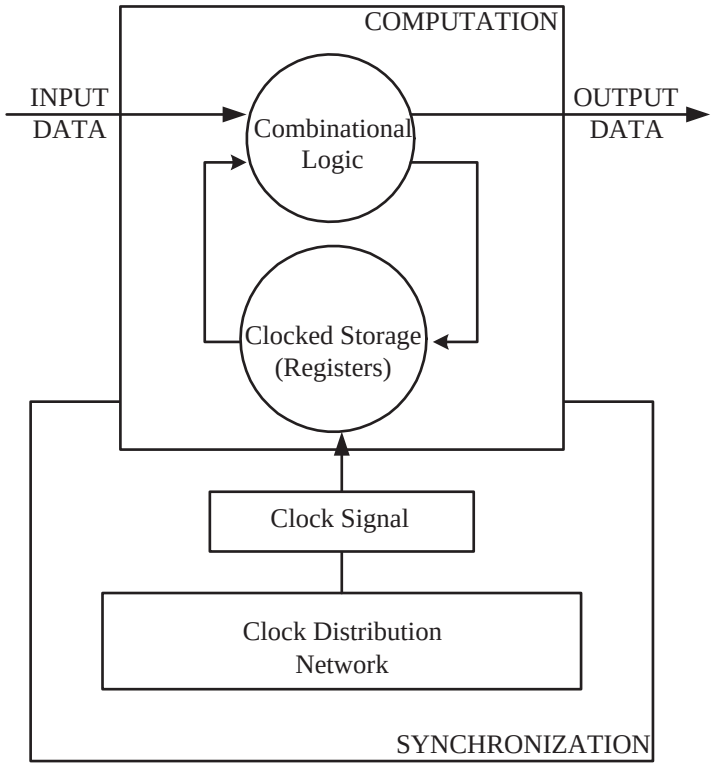


Figure 1: Finite state machine model of a synchronous system.

The operation of a synchronous system involves gradual propagation of data signals from the input pins towards the output pins of the circuit. The gradual advancement—and computation—of the data signal is orchestrated by the clock signal. Typically, the active level or the transition of the clock signal initiates the propagation of data from the input

terminals and storage elements towards the output terminals and next stages of storage elements. Each clock cycle constitutes a computational cycle, during which the data signals depart from their respective sources, are processed in the combinational logic between the registers and are finally stored in or delivered to the destination registers or the output terminals of the synchronous circuit, respectively.

In order to analyze the operational characteristics of a synchronous circuit, *local data paths* are defined. A local data path is the building block of a synchronous circuit and is formed by a collection of combinational logic gates (combinational logic block) between two clocked storage elements. The data signal is processed within a local data path once for each clock signal cycle. The data signal initiating from a storage element is processed in the combinational logic block and is stored in the next stage of storage elements, ready for the next clock cycle. Data propagation in some paths may take multiple clock cycles. These paths are called multi-cycle paths [71]. Multi-cycle paths are not specifically addressed in the scope of this work, however, the presented design and analysis methods can easily be modified to accommodate such paths.

Definition 1: Local data path. Let R_i and R_f be two registers in the clocked circuit network where the subscripts i and f stand for initial and final, respectively. Let the input and output terminals of these registers, and the signals present at these terminals be defined as shown in Figure 2. A local data path is the circuit architecture formed by a sequentially adjacent pair of registers [43] and the combinational logic block between them. The output Q_i of the initial register propagates through the combinational logic block, evaluating to

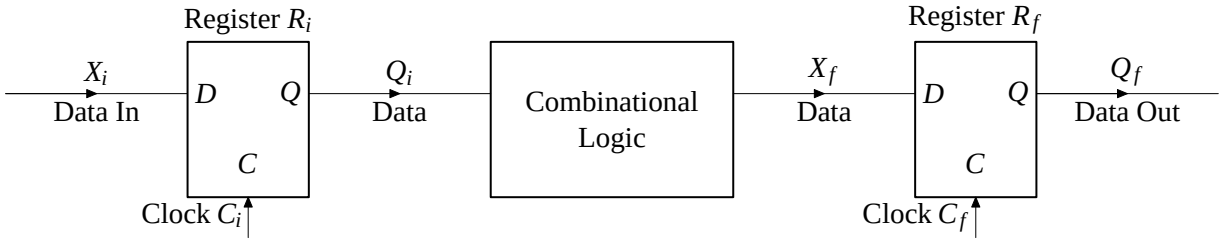


Figure 2: A local data path in a globally clocked synchronous circuit network.

the data signal X_f, before the data signal X_f arrives at the final register R_f. For proper operation of the sequentially adjacent pair of registers R_i and R_f, the data stored in R_i must be manipulated by the logic and stored into R_f during the next cycle of the clock signal C_f. Note that the clock signals C_i and C_f are representations for the two synchronization signals at the clock input terminals of registers R_i and R_f, respectively. The clock signals C_i and C_f are delivered with non-identical clock signal delays at respective registers, where these delays depend on the delivery path between the clock source and the destination register. The clock delivery paths constitute the clock distribution network.

The pair of delays $[D_{Pm}^{if}, D_{PM}^{if}]$ represents the minimum and maximum values, respectively, of the propagation time of the data signal on the combinational logic block on each local data path $R_i \rightarrow R_f$. This pair of minimum and maximum delay values for the logic block encompasses the combined effects of operating conditions and process parameter variations. The pair of delay values is provided as *parameters* to the timing analysis framework (and the CAD tool) presented in this dissertation by the specific delay estimation and signal integrity techniques that are used in the pre-analysis stage. In practice, such timing data is provided in standard design data files formats [71].

Note that the delay estimation process is *independent* of the timing analysis framework described in this dissertation. It is assumed throughout this dissertation that delay estimation and signal integrity techniques can accurately compute the changes in the timing characteristics of logic and synchronous components under various operating conditions. The inaccuracies of parameters (caused by imprecise delay estimation and signal integrity methods) that are passed on to the timing analysis framework are not investigated in the results of the presented work. However, designers must be aware of these dependencies.

The modeling and behavior of synchronous digital systems of interest are described in the rest of this chapter. Specifically, the operation of a synchronous system with level-sensitive latches is presented in Section 2.1. The representation of a synchronous circuit as a graph is described in Section 2.2. Generalized representations of single-phase and multi-phase synchronization schemes for synchronous circuits are defined in Section 2.3. In Section 2.4, the computer-aided design perspective adopted in completing this dissertation work is stated.

2.1 OPERATION OF A SYNCHRONOUS SYSTEM

Because of interconnect delays and process parameter variations on the clock distribution network, there may be significant time delays between the clock signals originating at the clock source and arriving at the destination storage elements of a digital synchronous circuit. The algebraic difference between the delays of the synchronizing clock signals at the initial and final register of a local data path is defined as *clock skew* [43]:

Definition 2: Clock Skew. Let R_i and R_f be a sequentially adjacent pair of registers (only combinational delay between registers) synchronized by the clock signals C_i and C_f, respectively. The clock skew between R_i and R_f is defined as

$$T_{skew}(i, f) = t_i - t_f, \tag{2.1}$$

where t_i and t_f are delays of the clock signals, C_i and C_f, from a common clock source to the registers R_i and R_f, respectively [43].

The clock skew is an algebraic difference which may evaluate to a negative, zero or positive value depending on the values of t_i and t_f. Positive clock skew has a limiting effect on the maximum operating frequency of a synchronous circuit [21, 43]. Negative clock skew on the other hand may effectively improve the minimum clock period of a circuit [21, 43]. Precise engineering of the clock tree network enables the utilization of negative skew on critical paths and permits positive clock skew on less-critical paths in order to improve the circuit performance. This approach is called *clock skew scheduling* [21].

Definition 3: Clock skew scheduling. Clock skew scheduling is a methodology to determine the optimal values of clock signal delays t_k to each register R_k in order to satisfy a design objective. Clock skew scheduling is typically performed to either minimize the clock period or to maximize the reliability of the circuit against secondary order effects on circuit operation. Frequently, the term *non-zero clock skew scheduling* is used to refer to *clock skew scheduling* [43].

Excessive negative and positive clock skew may lead to timing hazards in the circuit. Negative skew may cause data to be latched into the final register R_f during an earlier clock cycle than intended, thereby overwriting data latched during the earlier clock cycle. This

type of hazard is known as *double clocking* [43]. Similarly, positive skew may cause data to be lost by arriving late at the final register. This type of hazard is known as *zero clocking* [43]. The double clocking and zero clocking hazards are also called *hold* and *setup time violations*, respectively [74].

Data propagation in level-sensitive circuits is different compared to data propagation in edge-sensitive synchronous circuits due to the transparency property of latches. In level-sensitive circuits, the data signal arriving at a latch during the active level of the clock signal is immediately propagated through the latch. This fact leads to the phenomenon called *time borrowing*.

Definition 4: Time borrowing [15]. Time borrowing (also called cycle stealing [48]) refers to the time sharing phenomenon between consecutive clock cycles of adjacent local data paths due to the transparency of level-sensitive latches. Let $R_i \rightarrow R_j$ and $R_j \rightarrow R_k$ be two local data paths in a synchronous circuit S. Let S_{FF} and S_L denote the edge-sensitive (flip-flop-based) and level-sensitive (latch-based) synchronous circuits, respectively. In the edge-sensitive synchronous circuit S_{FF}, the larger of the maximum data propagation times on the local data paths $R_i \rightarrow R_j$ and $R_j \rightarrow R_k$ is the minimum clock period T_{FF}:

$$T_{FF} = \max\left(D_{PM}^{ij}, D_{PM}^{jk}\right) \tag{2.2}$$

In the level-sensitive circuit S_L, the transparency property of the latch R_j permits data propagation times higher than the minimum clock period T_L on the local data path $R_i \rightarrow R_j$ by borrowing time from the propagation on the next local data path—next clock cycle—$R_j \rightarrow R_k$:

$$T_L \leq \max\left(D_{PM}^{ij}, D_{PM}^{jk}\right) \tag{2.3}$$

It is possible to transform most edge-sensitive circuits into level-sensitive circuits with a simple procedure (the exceptions are some circuits with feedbacks). This procedure involves replacing the flip-flops in S_{FF} with latches and ensuring the proper delivery of clock signals at the latches in order to preserve the functionality of the circuit. Due to the preservation of the circuit topology with the modification procedure from an edge-sensitive circuit to a level-sensitive circuit (when applicable), a shorter minimum clock period $T_L \leq T_{FF}$ is feasible for a level-sensitive circuit S_L. This fact is illustrated in Figure 3, where the timing diagrams

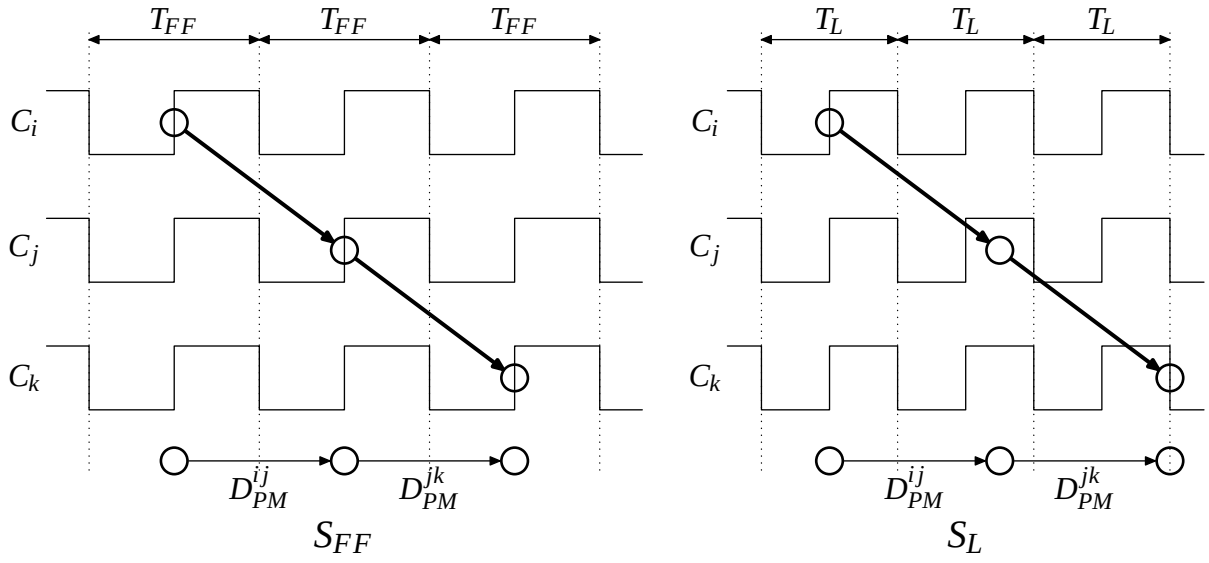


Figure 3: Effects of time borrowing on circuit operation.

for S_L and S_{FF} are shown on the left and right, respectively. In Figure 3, the variables D_{PM}^{ij} and D_{PM}^{jk} represent data propagation times on local data paths $R_i \rightarrow R_j$ and $R_j \rightarrow R_k$, respectively, and are represented by the one-sided arrows. Note that in the local data path $R_i \rightarrow R_j$ of S_L, the data signal arrival at R_j occurs during the transparent phase of R_j, borrowing time from the adjacent local data path $R_j \rightarrow R_k$. Detailed investigation of the time borrowing phenomenon is presented in [48].

2.2 GRAPH MODEL OF A SYNCHRONOUS SYSTEM

A graph model is often used for computer representation of a synchronous digital system. For convenience, the graph model representing a synchronous circuit, where each vertex represents a register and each edge represents a combinational logic block of a local data path, is called a circuit graph. A circuit graph provides a common abstract framework for the automated analysis of circuits.

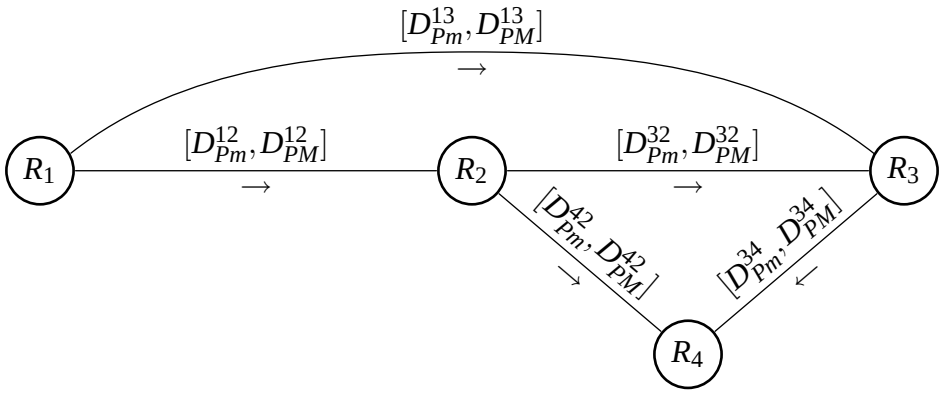


Figure 4: Circuit graph of a synchronous system with four registers and five data paths.

Definition 5: Circuit graph [43]. A fully synchronous digital circuit S is represented as a connected directed graph. The number of registers r and the number of data paths p in the synchronous circuit correspond to the number of vertices and the number of edges in the graph, respectively. The minimum and maximum data propagation times D_{Pm}^{if}, D_{PM}^{if}, respectively, are represented by an edge between the vertices corresponding to the registers R_i and R_f. The directed graph representation of a sample network is presented in Figure 4.

2.3 SYNCHRONIZATION SCHEMES

As mentioned in Section 2.1, the operation of a synchronous circuit is orchestrated by a globally distributed clock signal. The clock signal ensures the correct ordering of operations on local data paths. Clock signals are defined either with a *single-phase* or a *multi-phase* scheme.

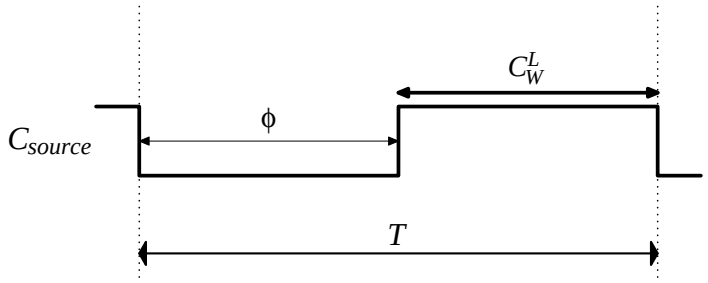


Figure 5: A generic single-phase synchronization clock.

2.3.1 Single-Phase Clock Synchronization

Single-phase (clock) synchronization is observed when the same clock signal is distributed to all the synchronous components of a circuit. Figure 5 presents a representation of a generic single-phase clock signal. The duty cycle is determined by the on-time of (C_W^L). The parameter C_{source} denotes the clock signal at the originating clock source. The subscripts to parameter C denote the clock signal at a destination register. For instance, the clock signal at an arbitrary register R_k is denoted C_k.

The relatively easy-to-implement and analyze single-phase scheme has several shortcomings in the synchronization of state-of-the-art VLSI circuits. In nano-scale CMOS implementations, the wire sizes shrink disproportionally with the feature size [35]. Thus, only a certain percentage of the chip is reachable during a single clock cycle [35].

2.3.2 Multi-Phase Clock Synchronization

Multi-phase (clock) synchronization is observed when different phases of the clock signal are distributed to the synchronous components of a circuit. Figure 6 presents a representation of a multi-phase clock signal. In Figure 6, the multi-phase synchronization scheme is generated with overlapping clock signal phases. In practical implementation, non-overlapping clock phases are used more frequently due to their practicality of implementation and analysis. The duty cycles of the clock phases are considered identical with on-times of (C_W^L).

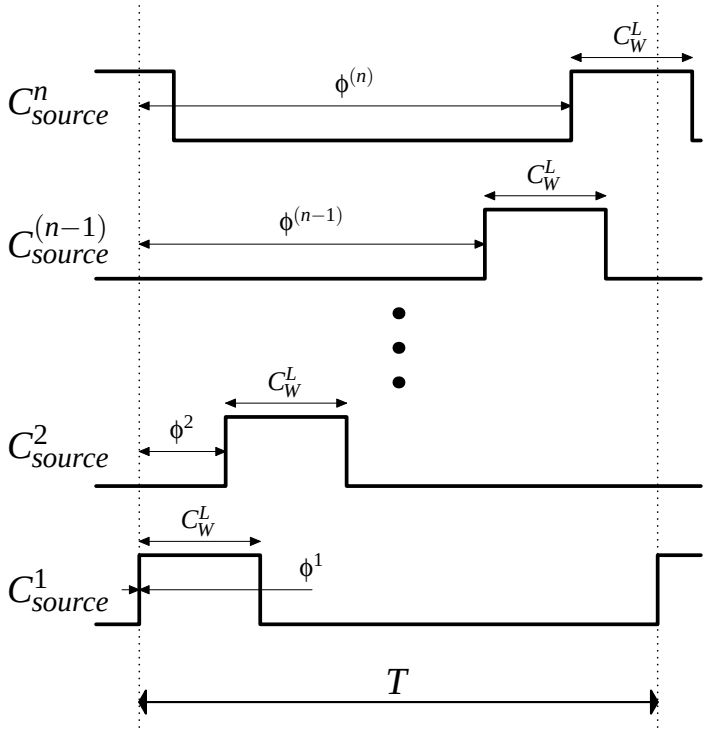


Figure 6: A generic multi-phase synchronization clock.

In Figure 6, the set of clock signals $C^{global} = \{C^1, \ldots, C^n\}$ constitutes the n-phase clocking scheme. The subscripts denote the location of the clock signals on the circuit. For instance, C^1_{source} denotes the clock signal at the clock source of the clock phase C^1. When this clock signal is delivered to an arbitrary register R_k, it is represented by C^1_k. The start time ϕ^{p_i} of clock signal phase C^{p_i} is defined with respect to a common reference clock cycle. The *phase shift operator* $\phi^{p_i p_f}$ [15] is used to transform variables between different clock phases. The phase shift operator $\phi^{p_i p_f}$ is defined as the algebraic difference $\phi^{p_i p_f} = \phi^{p_i} - \phi^{p_f} + kT$, where k is the number of clock cycles occurring between phases. Note that for a single-phase clocking scheme, the phase shift operator evaluates to $\phi^{if} = T$.

A multi-phase synchronization approach is advantageous in terms of increasing the reachability of circuit registers, creating less skew within physically neighboring local clock do-

mains and potentially saving power. Although multi-phase synchronization is advantageous in many aspects, the design and analysis of such synchronization schemes are more complex.

2.4 COMPUTER-AIDED DESIGN PERSPECTIVE

In this work, the design and timing analysis of circuits are presented primarily from a computer-aided-design (CAD) perspective. The timing analysis framework is developed with the assumption of the availability and controllability of multiple phases and fine-grained clock delays in both novel and conventional clocking technologies. The actual physical circuit implementation and other practical details of these clocking technologies are not the focus of this work. Comprehensive presentation and analysis of these clocking technologies can be found in various references such as [10, 11, 13, 23, 54, 55, 63, 96, 97]. Nevertheless, the operational characteristics of one of the target clocking technologies—the resonant rotary clocking technology—are presented in Chapter 7.

The resonant rotary clocking technology is described in order to emphasize the practical aspects of a synchronization scheme that can deliver controllable clock delays (hence, controllable clock skews) at high frequencies. The operational characteristics of a multi-phase clocking system (either provided by a novel clocking technology or by an improved conventional clock distribution system) are abstracted in order to develop the models for a CAD implementation.

Clock speed is the predominant measure for synchronous circuit performance. Consequently, this work uses the minimum clock period of a circuit as the figure of merit under various multi-phase synchronization schemes. Throughout this dissertation, the term *performance improvement* refers to improvement of the minimum clock period.

3.0 TIMING PROPERTIES OF REGISTERS

The general structure and principles of operation of a fully-synchronous digital VLSI system are described in Chapter 2. In an abstract overview, a synchronous circuit is identified by the storage elements (registers) and the synchronization scheme of the circuit.

Registers can be classified into two categories; (edge-triggered) flip-flops and (level-sensitive) latches. Flip-flops are sensitive to the changes in their data input terminals when the clock signal has low-to-high or high-to-low transitions while (level-sensitive) latches are sensitive when the clock signal has a certain level. In this chapter, the principles of operation for the two different types of storage elements—*flip-flops* and *latches*—are presented. In particular, the operation principles for edge-triggered flip-flops are discussed in Section 3.1 and the operation principles for level-sensitive latches are discussed in Section 3.2.

3.1 PARAMETERS OF AN EDGE-TRIGGERED FLIP-FLOP

The specific circuit design or electrical implementation of an edge-triggered flip-flop need not be considered in this work. At a higher level of abstraction, the timing properties of flip-flops are encapsulated by certain timing parameters. These parameters connect the events on the input, output and clock terminals of a flip-flop.

A *flip-flop* is a type of register which is sensitive to the transition of the synchronizing clock signal. Accordingly, a *flip-flop* is commonly referred to as an edge-triggered flip-flop or edge-triggered register. A typical edge-triggered flip-flop with a clock signal C, input signal D and output signal Q is shown in Figure 7. The operation of a flip-flop is presented in Figure 8. Note that the presented flip-flop is a positive-edge triggered flip-flop, whose data

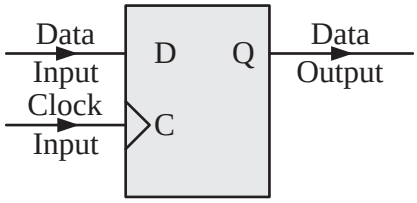


Figure 7: An edge-triggered flip-flop or register symbol.

output latches the input signal when the clock signal makes a low-to-high transition. The sensitive region for a flip-flop, where the register latches the data, is indicated by the shaded region in Figure 8.

A typical cycle of a clock signal synchronizing a positive edge-sensitive flip-flop is shown in Figure 9. The length of the clock period is denoted by the parameter T. The on-time is represented by the parameter C_W^L. In Figure 9, the triggering edge occurs at time t_3. As pointed out in Section 2.2, the data propagation time through the combinational logic block

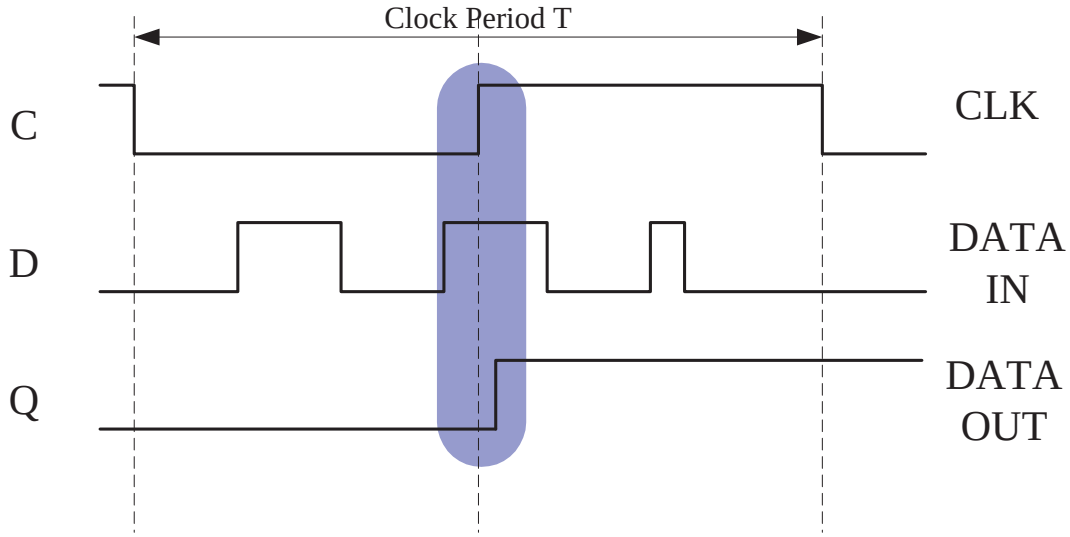


Figure 8: Typical operation of an edge-triggered flip-flop shown in Figure 7.

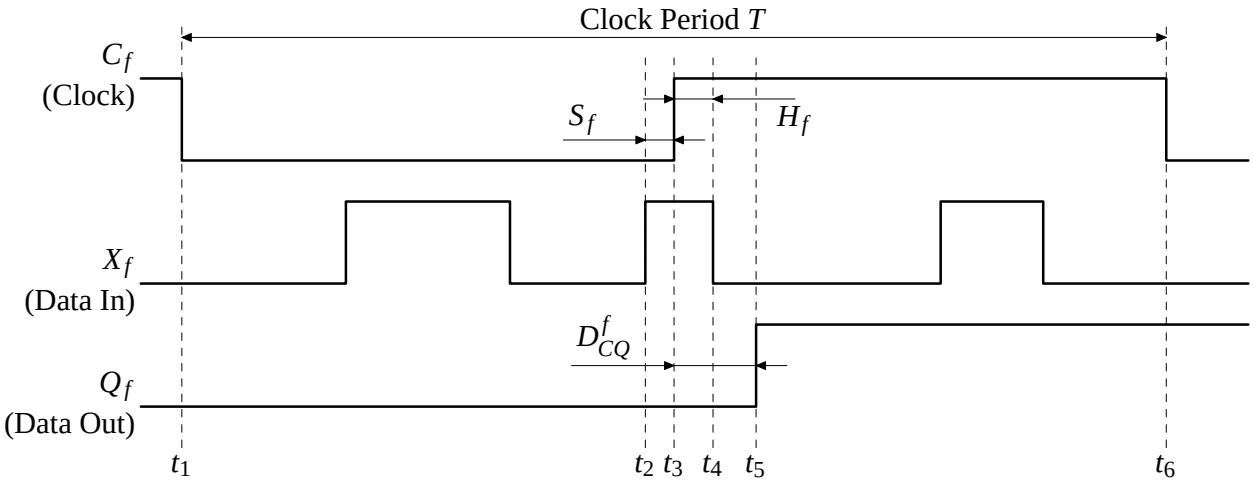


Figure 9: Timing properties of a flip-flop in a circuit with a clock period T.

of a local data path $R_i \rightarrow R_f$ is defined $\left[D^{if}_{Pm}, D^{if}_{PM}\right]$. The subscripts m and M stand for the minimum and maximum values, respectively. In Figure 9, the operation of the final flip-flop R_f of a local data path is illustrated.

Parameters H_f, S_f, D_{CQ} and C^L_W refer to the hold time, setup time, clock-to-output delay and clock on-time, respectively. Hold time is the minimum time that the data signal D must remain stable after the triggering edge of the clock signal so that it is registered during the intended clock cycle. In Figure 9, the value $H_f = t_4 - t_3$ labels the hold time for the given clock cycle on R_f. Setup time is the minimum time between the triggering edge of the respective clock cycle and a change in X_f such that the new data value can be registered during the intended clock cycle. The setup time on R_f is illustrated with $S_f = t_3 - t_2$. The propagation delay of the data signal from the input terminal to the output terminal after the active transition of the clock signal—clock-to-output delay—is shown as $D_{CQ} = t_5 - t_3$. The subscripts m and M appended to the parameter D_{CQ} stand for the minimum and maximum delay values, respectively.

3.2 PARAMETERS OF A LEVEL-SENSITIVE LATCH

Similar to the edge-sensitive flip-flops discussed in Section 3.1, the specific circuit design or electrical implementation of a level-sensitive latch is not considered in this work. At a higher level of abstraction, the timing properties of latches are encapsulated by certain timing parameters. These parameters connect the events on the input, output and clock terminals of a level-sensitive latch.

A *latch* is a type of register which is sensitive to the level of synchronizing clock signal. Accordingly, a *latch* is commonly referred to as a level-sensitive latch or level-sensitive register. A typical level-sensitive latch with a clock signal C, input signal D and output signal Q is shown in Figure 10. The operation of the level-sensitive latch is presented in Figure 11. Note that the presented latch is a positive-level sensitive latch, whose data output follows any change in the input signal when the clock signal remains at its positive valued. In level-sensitive circuits, the active level of the synchronizing clock signal defines the *transparent* state (or transparent phase) and the inactive level of the clock signal defines the *opaque* state (or opaque phase) of latch operation. The transition of the clock signal which starts the transparent state is called the *leading* edge, while the *trailing* edge is the transition of the clock signal which concludes the transparent state and marks the beginning of the opaque phase. In Figure 11, the transparent state is indicated by the shaded region on the timing diagram.

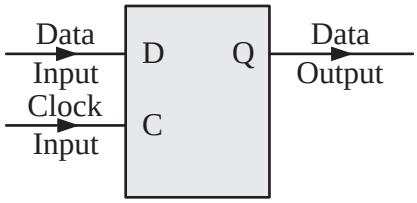


Figure 10: A level-sensitive latch or register symbol.

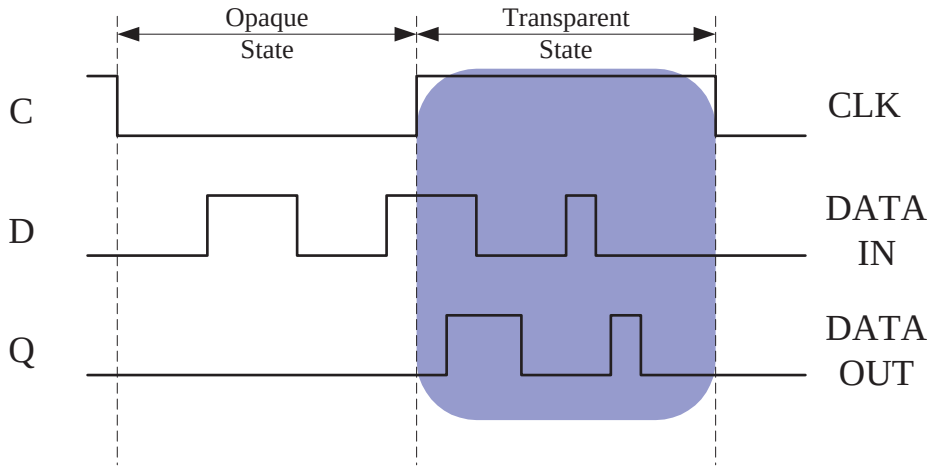


Figure 11: Typical operation of a level-sensitive latch shown in Figure 10.

Without affecting the generality of the presented work, the formulation of the timing constraints is derived for a specific reference clock cycle. The reference clock cycle can be selected as starting with the inactive value of the clock signal followed by the active value (opaque-phase-first) or vice versa (transparent-phase-first). In this work, the timing constraints are formulated considering an opaque-phase-first clock signal driving positive level-sensitive latches. A typical cycle of such a clock signal synchronizing a positive level-sensitive latch R_f is shown in Figure 12. The length of the clock period is denoted by the parameter T. The minimum time interval between the leading and trailing edges of the clock signal—commonly called the on-time and defining the transparent phase—is represented by the parameter C_W^L. In Figure 12, the *leading* and *trailing* edges occur at times t_3 and t_8, respectively.

The data propagation time D_P^{if} through the combinational logic block of a local data path $R_i \rightarrow R_f$ was defined in Section 2.2. The propagation of the data signal is formulated during two consecutive clock cycles, generically called the k-th and $(k+1)$-th. The data signal departs from R_i during the k-th clock cycle, is processed in the combinational logic block and arrives at the destination register R_f during the $(k+1)$-th clock cycle. Note that the departure of the data signal from R_i occurs during the transparent phase of the k-th

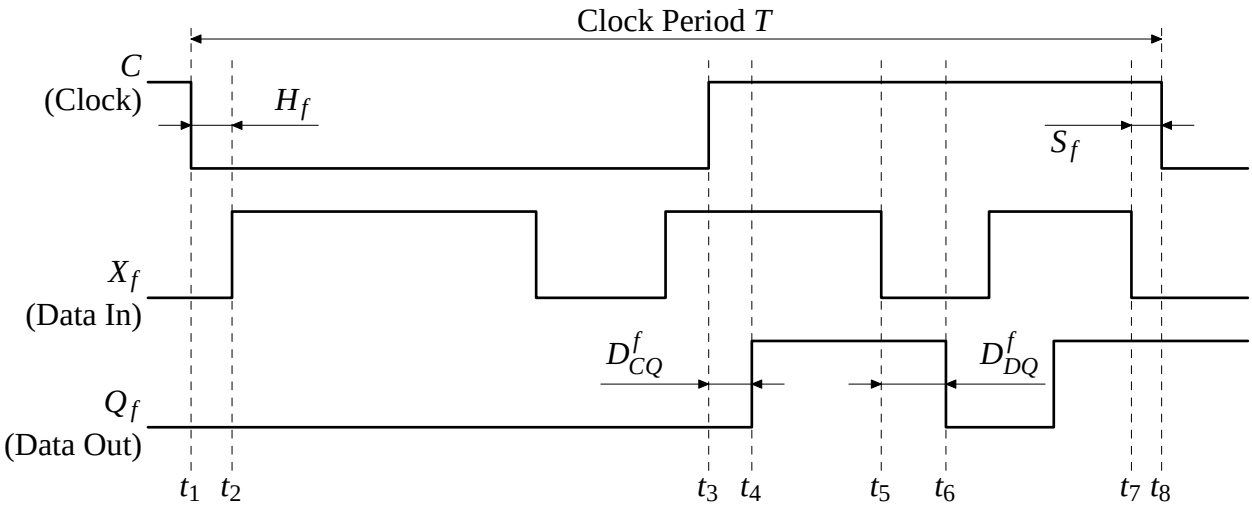


Figure 12: Timing properties of a level-sensitive latch in a circuit with a clock period T.

cycle. The arrival of the data signal at R_f can occur both during the opaque and transparent phases of the $(k+1)$-th cycle. If the data signal arrives during the transparent phase, it is immediately propagated through the latch R_f. If the data signal arrives during the opaque phase of R_f, the data signal has to remain stable until the leading edge of the clock signal (beginning of the transparent phase) to propagate through the latch R_f.

Parameters H_f, S_f, D_{DQ}, D_{CQ} and C^L_W which stand for the hold time, setup time, data-to-output delay, clock-to-output delay and clock on-time, respectively are introduced for latches. Hold time is the minimum time that the data signal D must remain stable after the trailing edge of the clock signal so that it is latched during the intended clock cycle. In Figure 12, the value $H_f = t_2 - t_1$ labels the hold time for the given clock cycle on the final register R_f of a local data path. Setup time is the minimum time between the trailing edge of the respective clock cycle and a change in X_f such that the new data value can be latched in the intended clock cycle. The setup time on R_f is illustrated with $S_f = t_8 - t_7$. The propagation delay of the latch from the data input terminal to the output terminal—data-to-output delay—on R_f is shown as $D_{DQ} = t_6 - t_5$. The propagation delay of the latch from the clock input terminal to the output terminal—clock-to-output delay—is shown as

$D_{CQ} = t_4 - t_3$. The subscripts m and M appended to the parameters D_{DQ} and D_{CQ} stand for the minimum and maximum delay values, respectively.

4.0 STATIC TIMING ANALYSIS OF LEVEL-SENSITIVE CIRCUITS

Static timing analysis is currently the prevailing methodology to verify the temporal correctness of integrated circuits due to its accuracy, relatively low computational complexity and the practicality of accompanying timing models. One or more of these advantages of static timing analysis can be impaired in the analysis of complex design structures or sophisticated synchronization schemes. One computationally hard situation, for instance, is analyzed in this chapter; namely, *the static timing analysis of level-sensitive synchronous circuits*.

Level-sensitive (latch-based) circuits are gaining popularity in the state-of-the-art high-performance synchronous circuit design due to their smaller size, lower power consumption and faster operation speeds [53, 88, 89]. The timing analysis of level-sensitive circuits, however, is more difficult due to the non-linearity of the timing constraints caused by the time borrowing property [15]. Traditionally, the non-linearity of constraints has been resolved with one of two approaches. On one hand, analyses which aim to accurately model the effects of time borrowing have been considered too optimistic and this property is fully disregarded from the analysis [15, 74]. More recently, the non-linear constraints of operation are relaxed using iterative solution techniques [8, 64, 73, 99]. The iterative solution techniques are practical. However, these techniques are problem-specific. In this chapter, a novel linear programming (LP) formulation applicable to the timing analysis of large-scale level-sensitive synchronous circuits is presented. The presented LP formulation accurately models the effects of time borrowing. This LP formulation is computationally efficient due to the linearization of non-linear constraints, and the formulation and solution processes are fully-automated.

Digital VLSI synchronous circuits are subject to different types of timing analyses. Traditional among these analysis are three different problems: clock period minimization [8, 15, 21,

42, 43, 64, 74, 77, 78], clock period verification [15, 73, 99] and circuit retiming [46, 49, 51, 69]. Clock period minimization is the analysis of a synchronous circuit in order to solve for the minimum clock period—the maximum operating frequency—of a synchronous circuit. Clock period verification is the analysis to ensure that a synchronous circuit is fully-operational for a given clock period. Circuit retiming is the analysis of a synchronous circuit aiming to achieve higher operating frequencies by modifying the circuit network.

Even though there are different types of timing analysis problems, the operation of the synchronous circuit under scrutiny is identical in all cases (possibly except for retiming problems). Thus, in the formulation of the timing analysis problem, a framework of constraints identifying synchronous circuit operation is essential. The categorized set of constraints are verified at each local data path of a circuit subject to a specific objective function, constructing the static timing analysis.

The generation of a general framework for the timing analysis of level-sensitive circuits is also discussed in this chapter. The framework is used to formulate and solve the clock skew scheduling problem of level-sensitive circuits. The clock skew scheduling problem of edge-sensitive synchronous circuits has previously been addressed with both linear and non-linear formulations in several publications [2, 21, 30, 43, 60]. The clock skew scheduling problem of level-sensitive circuits, however, has been unexplored. Simultaneous with the development of the timing analysis framework for level-sensitive circuits, the clock skew scheduling problem of level-sensitive circuits is formulated as an LP problem.

In Section 4.1, the *operational constraints* governing non-zero clock skew, level-sensitive synchronous circuit operation are introduced. In Section 4.2, a brief overview of previously offered algorithms for the clock period minimization problem of zero clock skew, level-sensitive circuits is presented. The *constructional constraints* defined for the novel LP model clock period minimization problem are introduced in Section 4.3. In Section 4.4, linearization of the clock period minimization problem is described. In Section 4.5, the proposed solution method is applied to a set of benchmark circuits for experimentation. In Section 4.6, the optimality of the LP model problem formulation is discussed. Experimental results are analyzed in Section 4.7. Additional observations and potential enhancements to the proposed formulation are offered in Section 4.8. Conclusions are offered in Section 4.9.

4.1 OPERATIONAL TIMING CONSTRAINTS

Certain conditions must be satisfied for every sequentially adjacent pair of registers in a level-sensitive synchronous circuit in order to prevent timing hazards. These conditions are encapsulated by four sets of operational constraints and two sets of constructional constraints. The *operational constraints* are the constraints that model the operation of a level-sensitive synchronous circuit. The *constructional constraints* are defined to ensure the correctness and completeness of the formulation of the proposed timing analysis problem. The definitions for the first three sets of operational constraints—called *latching*, *synchronization* and *propagation* constraints, respectively—are derived from the zero clock skew definitions in [15]. The fourth set of operational constraints—called the *skew* constraints—are derived from the skew definitions for edge-sensitive synchronous circuits presented in [43]. The latching, synchronization, propagation and skew constraints for a single-phase synchronization system are described in Sections 4.1.1 , 4.1.2 , 4.1.3 and 4.1.4, respectively.

4.1.1 Latching Constraints

Latching constraints bound the arrival time of the data signal X_f (recall the local data path in Figure 2 on page 8) in order to ensure that X_f is latched during the intended clock cycle. The earliest arrival time of X_f at the data input terminal of R_f is denoted by the parameter a_f. Similarly, the latest arrival time of X_f is denoted by A_f. Both parameters are defined in the frame of reference of the native clock cycle, that is, relative to the beginning of the current clock cycle.

The interval for the data arrival time is characterized by the hold time and the setup time requirements of R_f as follows:

$$H_f \leq a_f \tag{4.1}$$

$$A_f \leq T - S_f. \tag{4.2}$$

Eq. (4.1) constrains the earliest arrival of X_f at R_f. The earliest data arrival time must be no earlier than hold time after the trailing edge (t_2 in Figure 12) of the previous clock cycle.

Suppose the $(k+1)$-th clock cycle at latch R_f is illustrated in Figure 12 on page 22, where $t_1 = t_f + kT$ [zero in the frame of reference of $(k+1)$-th cycle]. The hold time is defined by the difference $t_2 - t_1$. If data arrives at R_f earlier than the hold time, a double-clocking hazard occurs.

Similarly, (4.2) represents the setup constraint on R_f. As shown in Figure 12, the data must arrive at the final latch at least setup time prior to the trailing edge of the clock cycle. Assuming the $(k+1)$-th clock cycle is illustrated in Figure 12, the trailing edge of the clock cycle occurs at $t_8 = t_f + (k+1)T$ [T in the frame of reference of the $(k+1)$-th cycle]. Thus, data cannot be latched into R_f during the $(k+1)$-th cycle if the data arrives later than $t_7 = t_f + (k+1)T - S_f$ [$(T - S_f)$ in the frame of reference of the $(k+1)$-th cycle]. Late arrival of the data signal results in a zero clocking hazard as previously described in Chapter 2.

4.1.2 Synchronization Constraints

Synchronization constraints define the departure time of the data signal Q_i from the initial latch of a local data path. The departure time from a latch depends on the state of the latch—transparent or opaque. Implementation-specific register internal delays, D_{DQ} and D_{CQ}, affect the departure times in transparent and opaque states of operation, respectively. The earliest departure time d_i of Q_i from R_i is defined in (4.3). The latest departure time D_i is defined by (4.4):

$$d_i = \max\left(a_i + D^i_{DQm}, T - C^L_W + D^i_{CQm}\right), \tag{4.3}$$

$$D_i = \max\left(A_i + D^i_{DQM}, T - C^L_W + D^i_{CQM}\right). \tag{4.4}$$

An exhaustive inspection of all possible cases of earliest and latest departure times during the k-th clock cycle is shown in Figure 13. The time intervals for the arrival and departure times are illustrated by the upper and lower parallel dotted lines, respectively. The left and right ends of these dotted lines in the figure correspond to earliest and latest times, respectively. The lengths of the white and black rectangular boxes correspond to the clock-to-output and data-to-output latch delays, respectively. Note that cases V through VIII may exhibit timing hazards.

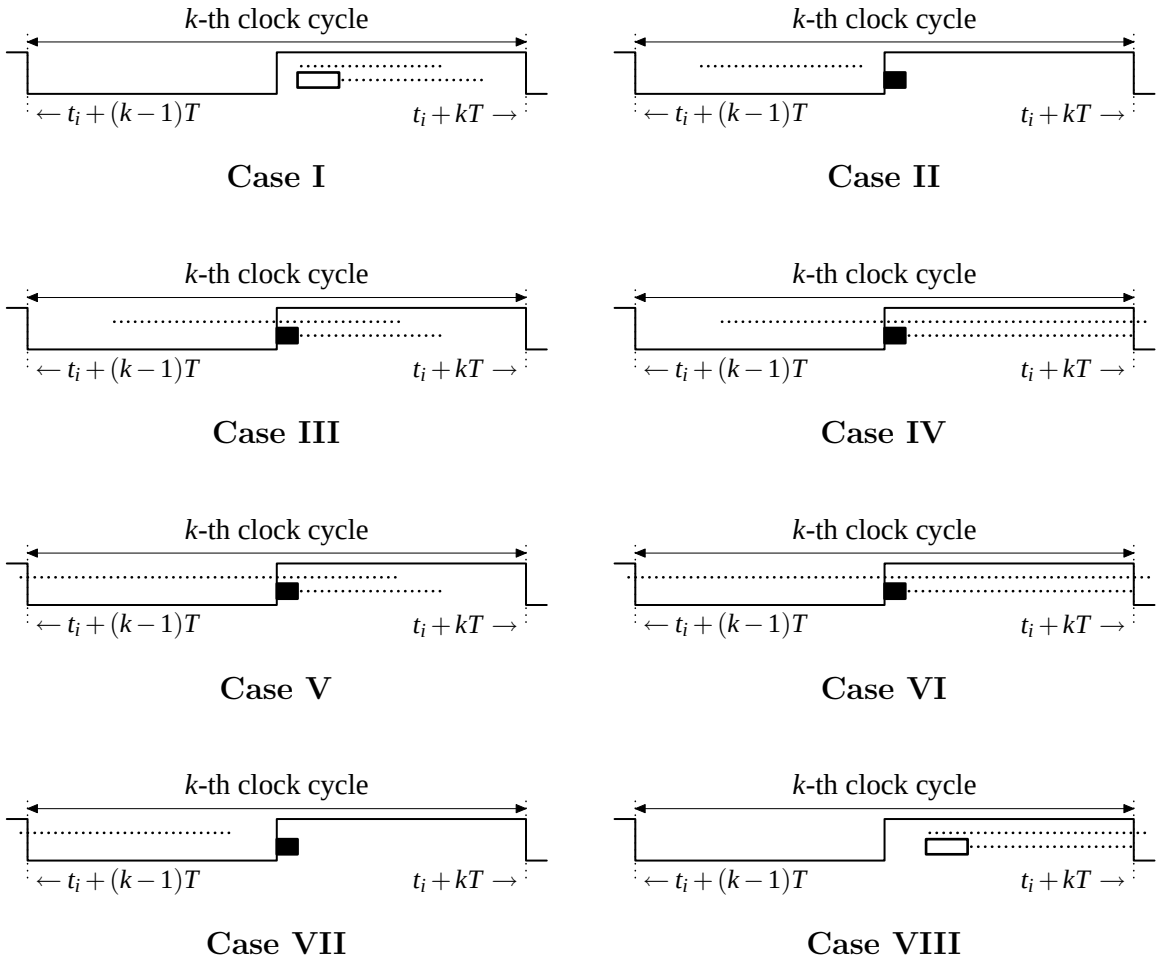


Figure 13: Possible cases for the arrival and departure times of data at the initial latch.

Consider (4.3), which describes the earliest departure time of the data signal Q_i from latch R_i. The first term of the max function, $\left(a_i + D^i_{DQm}\right)$, describes the time instant when the input data arrival occurs at its earliest time during the active phase of the clock signal C_i. The data signal immediately propagates through the latch (as illustrated in cases I and VIII of Figure 13). In these cases, the earliest departure time d_i from R_i depends on the earliest arrival time a_i of the data signal and the time D^i_{DQ} it takes for the data to appear at the output terminal of R_i.

The second term of the max function, $\left(T - C^L_W + D^i_{CQm}\right)$, refers to the case when the earliest data arrival time occurs during the opaque phase of R_i. In the opaque phase of

operation, the departure time of the data signal from the initial latch occurs clock-to-output delay D^i_{CQ} later than the leading edge of the clock signal. Such data propagation is illustrated in cases II-VII of Figure 13. The max function is used to combine these cases and to define the earliest departure time d_i from the initial latch R_i. Similar reasoning applies to the derivation of the latest departure time D_i defined by (4.4).

4.1.3 Propagation Constraints

Propagation constraints define the arrival time of the data signal X_f at the final latch R_f of a local data path. These constraints are as follows:

$$a_f = \left(\min_i \left[d_i + D^{if}_{Pm} + T_{skew}(i,f)\right]\right) - T \tag{4.5}$$

$$A_f = \left(\max_i \left[D_i + D^{if}_{PM} + T_{skew}(i,f)\right]\right) - T. \tag{4.6}$$

For each incoming path to latch R_f, the lower bound for a_f is individually calculated using the expression $\left[d_i + D^{if}_{Pm} + T_{skew}(i,f) - T\right]$. The minimum of the arrival times among the incoming data paths is assigned as the earliest arrival time at R_f. The latest arrival time A_f for the data signal is defined similarly. In case of multiple data paths fanning into R_f, the maximum of the arrival times among the incoming data paths is the latest arrival time of the data signal at R_f. These two facts are implied in the formulation by the inclusion of the min and max functions in (4.5) and (4.6), respectively.

The propagation constraints are illustrated on a sample synchronous circuit in Figure 14. Note that in Figure 14, two local data paths starting at the latches R_{i_1} and R_{i_2} and ending at R_f are considered. The time intervals for the arrival and departure times of the data signal are illustrated by the upper and lower parallel dotted lines, respectively. The lengths of the white and black rectangular boxes correspond to the clock-to-output and data-to-output latch delays, respectively. The earliest arrival time is illustrated on the data path $R_{i_1} \rightarrow R_f$. The data signal departs from R_{i_1} at time d_{i_1} and propagates on the data path $R_{i_1} \rightarrow R_f$ for a time period of $D^{i_1 f}_{Pm}$. The earliest data arrival time $\left[d_{i_1} + D^{i_1 f}_{Pm} + T_{skew}(i_1,f) - T\right]$ observed on this data path is earlier than the arrival time $\left[d_{i_2} + D^{i_2 f}_{Pm} + T_{skew}(i_2,f) - T\right]$ observed on the only other incoming path to R_f, $R_{i_2} \rightarrow R_f$. Hence, the earliest data arrival time a_f at

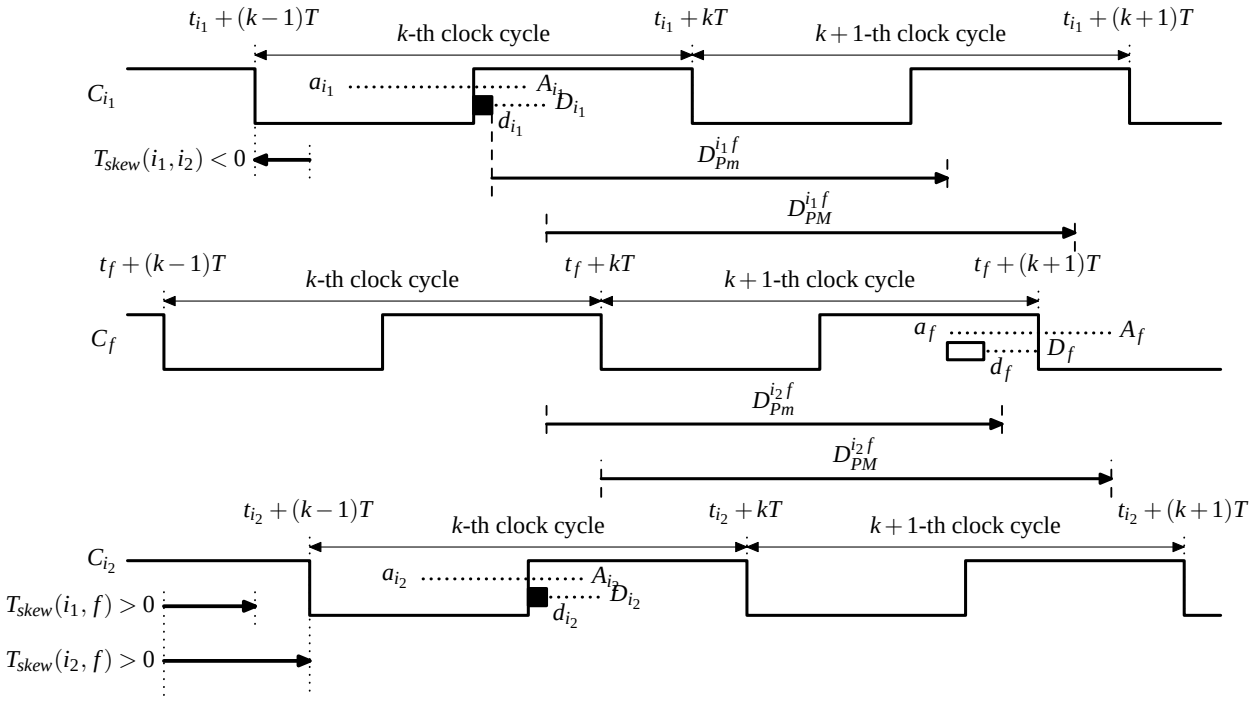


Figure 14: Propagation of the data signal in a simple circuit.

R_f is defined by the propagation on the $R_{i_1} \rightarrow R_f$ data path. Similarly, on the data path $R_{i_2} \rightarrow R_f$, a maximum data propagation time of $D_{PM}^{i_2 f}$ elapses conferring the latest data arrival time at R_f, $\left[A_f = D_{i_2} + D_{PM}^{i_2 f} + T_{skew}(i_2, f) - T\right]$.

The departure of Q_i and the arrival of X_f must occur during two consecutive clock cycles for proper circuit operation. In order to switch between the frame of references of these two cycles, the phase shift operator ϕ^{if} is used. The phase shift operator evaluates to $\phi^{if} = T$ for single-phase synchronization as discussed in Section 2.3. Thus, the clock period T is subtracted from the calculated arrival time in order to shift the point of reference of the data arrival time at R_f to the beginning of the previous clock cycle.

4.1.4 Skew Constraints

Skew constraints introduce lower and upper bounds on clock skew on a local data path:

$$A_i + D^i_{DQM} + D^{if}_{PM} \leq 2T - T_{skew}(i,f) - S_f \tag{4.7}$$

$$D^i_{CQM} + D^{if}_{PM} \leq T + C^L_W - T_{skew}(i,f) - S_f \tag{4.8}$$

$$\max\left[a_i + D^i_{DQm}, T - C^L_W + D^i_{CQm}\right] + D^{if}_{Pm} \geq T - T_{skew}(i,f) + H_f. \tag{4.9}$$

Presence of clock skew in level-sensitive synchronous circuits significantly affects the system timing. The latching [(4.1) and (4.2)], synchronization [(4.3) and (4.4)] and propagation [(4.5) and (4.6)] constraints presented previously are derived considering the presence of non-zero clock skew in the clock tree network. These three sets of constraints naturally impose lower and upper bounds on clock skew. Thus, the skew constraints are redundant if a typical minimization of the clock period problem is pursued. However, if implementation-specific constraints modify or suppress any of the given constraints, such that, the bounds on clock skew are invalidated, the skew constraints are essential to the correct analysis of the circuit.

The effects of clock skew on synchronous circuit operation can be derived from the latching [(4.1) and (4.2)], synchronization [(4.3) and (4.4)] and propagation [(4.5) and (4.6)] constraints. Note that the variable A_f described in (4.2) can be expressed as follows:

$$A_f = D_i + D^{if}_{PM} + T_{skew}(i,f) - T. \tag{4.10}$$

Substituting (4.4) in (4.10), then substituting the result into (4.2) leads to,

$$\max\left[A_i + D^i_{DQm}, T - C^L_W + D^i_{CQm}\right] + D^{if}_{PM} + T_{skew}(i,f) - T \leq T - S_f, \tag{4.11}$$

conveniently represented by the first two sets of skew constraints [(4.7) and (4.8)].

Eq. (4.1) must hold to prevent the early arrival of data signal, where a_f depends on d_i as implied by (4.5):

$$a_f = d_i + D^{if}_{Pm} + T_{skew}(i,f) - T. \tag{4.12}$$

Eq. (4.12) also depends upon whether the data signal arrives before or during the transparent state of the latch. Substituting (4.3) into (4.12), (4.12) into (4.1) and rearranging the terms

lead to the last set of the skew constraints, (4.9). Eq. (4.9), re-written in (4.13), is a non-linear skew constraint, as the elimination of the max function is not straightforward:

$$H_f \leq \max\left[a_i + D^i_{DQm}, T - C^L_W + D^i_{CQm}\right] + D^{if}_{Pm} + T_{skew}(i,f) - T. \tag{4.13}$$

As stated before, the skew constraints [(4.7), (4.8) and (4.9)] are redundant in the formulation of a typical clock period minimization problem, as these constraints are derived from the existing set of constraints [(4.1)–(4.6)]. The skew constraints are not included in the LP model presented in Section 4.4.2, but are used in the verification of the proposed solution method in Section 4.7.

4.2 ITERATIVE APPROACH TO STATIC TIMING ANALYSIS

The operational constraints (Section 4.1) provide a system of equations defining the timing operation of a level-sensitive synchronous circuit. Different versions of the constraints presented in Section 4.1 have been used by designers in order to develop timing analysis models for zero clock skew, level-sensitive circuits. The set of constraints initially defined for the clock period minimization problem of a conventional zero clock skew problem in [15] is known as the *SMO formulation* [65].

A popular timing analysis approach for level-sensitive circuits is presented in [8, 64, 65, 74] based on the SMO formulation. This timing analysis approach involves several algorithms targeting clock period verification and minimization problems, all based on the analytical framework described in Section 4.1. The proposed algorithms are iterative algorithms. In particular, very small values are assigned to the timing variables of a circuit and the circuit is investigated for timing violations by iteratively incrementing the values of the timing variables. It is important to note that the clock delay values t_i and consequently the clock skew values $T_{skew}(i,f)$ are pre-determined numerical values in these algorithms. Thus, these *iteration-based algorithms do not support clock skew scheduling.*

The iterative algorithm proposed in [64] for the clock period minimization problem of level-sensitive circuits is presented in Figure 15. In the algorithm, r is the number of registers

```
//Initialize the latch arrival times
for i = 1 to |r| {
    A_i^prev = a_i^prev = −∞;
    // iterate the evaluation of the departure and arrival time equations
    // until convergence or a maximum of |r| iterations
    iter = 0;
    repeat
        iter = iter + 1;
        // update the latch departure times based on the latch arrival times
        // computed in the previous iteration
        for i = 1 to |r| {
            D_i = max ( A_i^prev, φ_i + D_i );
            d_i = max ( a_i^Prev, φ_i + d_i );
        };
        // update the latch arrival times based on the just-computed
        // latch departure times
        for i = 1 to |r| {
            A_i = max_j ( D_j + D_PM );
            a_i = min_j ( d_j + D_Pm );
        };
    until ( ( ( A_i = A_i^prev ) && ( a_i = a_i^prev ) ) || ( iter + 1 > |r| ) ) ) ;
};
// check and record setup and hold violations
for i = 1 to |r| {
    SetupVio[i] = A_i > T - S_i + d_i;
    HoldVio[i] = a_i < H_i + D_i;
};
```

Figure 15: The iterative algorithm for static timing analysis of level-sensitive circuits.

in the synchronous circuit. The a, d, A and D vectors are the earliest arrival/departure and latest arrival/departure times, respectively, where the superscript $prev$ identifies the value of a variable in the previous clock cycle. The variables $SetupVio$ and $HoldVio$ hold the timing violation information for each register. In this algorithm, the arrival times are initialized to $a_i = A_i = -\infty$, where the algorithm simulates the start-up timing of the circuit. At each iteration step, the execution of the circuit at a clock cycle is simulated. Finally, once the arrival and departure times of the latches are determined, the algorithm checks for potential setup and hold time violations.

The algorithm presented in Figure 15 has been shown to converge to solutions relatively quickly [64]. The algorithm complexity is reported as $O(|r||p|)$, where $|r|$ is the number of latches in a circuit and $|p|$ is the number of edges of a circuit graph (recall from Section 2.2 that the number of edges of a circuit graph is the number of local data paths). However, it has been proven in [74] that in case of data-path loops (sequential feedback) in the synchronous circuit, the arrival and departure times might increase without bound. This leads to a setup violation and the described algorithm fails to provide reasonable run times. In [8], a correction is offered to the algorithm. This correction is based on the assumption that, a data path loop in the circuit can be detected in $|r|$ iterations. Thus, the algorithm is modified to artificially limit the number of iteration steps by $|r|$. In the modified algorithm, the complexity of the resulting algorithm is cubic in the number of registers r, as each iteration involves examining up to $|p|$ edges, and p is at most $|r|^2$.

The iterative algorithm presented in Figure 15 is later modified in [29] and [99] in order to account for multiple clock domains and crosstalk, respectively. Briefly, even though the iterative algorithm provides an initial and useful formulation for the timing analysis of level-sensitive circuits, the algorithm may fail in the presence of data path loops and does not provide a common framework for general timing analysis. In the next two sections, a novel model for the timing analysis of level-sensitive synchronous circuits is developed. The developed model constitutes a well-defined framework for general timing analysis problems.

4.3 CONSTRUCTIONAL TIMING CONSTRAINTS

The constructional constraints, called *validity* and *initialization* constraints, are defined to ensure the correctness and completeness of the formulation of the presented timing analysis problem. The first type of constructional constraints, the validity constraints, are presented in Section 4.3.1. The second type of constructional constraints, the initialization constraints, are presented in Section 4.3.2.

4.3.1 Validity Constraints

The definitions of the parameters a_f, A_f, d_f and D_f require the value of a_f (d_f) to be smaller than or equal to the value of A_f (D_f):

$$A_f \geq a_f \tag{4.14}$$

$$D_f \geq d_f. \tag{4.15}$$

While the four sets of operational constraints introduced in the preceding sections model the timing properties of the circuit, the required *sequentiality in time* of the referred variables is not explicitly enforced. Consistency in the definitions of a_f, A_f, d_f and D_f, must be maintained through post-solution checks or by including additional constraints. A solution leading to a result where $a_f > A_f$, for instance, is incorrect and must be disregarded.

Introducing the validity constraints [(4.14) and (4.15)] in the problem formulation is preferred over performing post-solution checks for two significant reasons. The first reason is to gain the ability to easily detect the feasibility of the problem. The second reason is to preserve the automation of the solution procedure.

4.3.2 Initialization Constraints

The LP model clock skew scheduling problem is formulated in order to minimize the clock period of a synchronous circuit. Besides the minimum clock period, it may also prove essential to accurately calculate the nominal data arrival and departure times for each register. The initialization constraints are introduced in order to fulfill this purpose, by leading to a

consistent timing schedule for the data signal propagation in a level-sensitive synchronous circuit.

After clock skew scheduling, the feasible (or optimal) solution set for one or more variables can be a range of values rather than a specific value. For instance, suppose that the earliest arrival time of a data signal at an arbitrary latch R_k can get any value in the interval $1.8 \leq a_k \leq 2.3$ without changing the minimum clock period of the circuit. For consistency, it is preferable to assign the smallest value to the earliest arrival time ($a_k = 1.8$). In general, it is better to assign the smallest possible values to the earliest arrival and departure time variables and the largest possible values to the latest arrival and departure time variables (where applicable). Such assignment provides a more comprehensive representation of data propagation (and sensitivity information [92]) in the system. Identification of the sensitivity information is useful to check for the consistency of the timing schedule generated by the LP problem (if necessary) as will be briefly discussed in Section 4.7.

Note that, the earliest and latest data arrival times at all registers, except for the *input* registers, are set to their lowest and highest possible values, respectively. These assignments are enforced by the propagation constraints [(4.5) and (4.6)]. The values assigned to the earliest and latest data arrival times (a, A) at the input registers do not affect the minimum clock period unless the assigned values cause the departure times to change. It may even be considered redundant to define earliest and latest arrival time variables (a, A) at the input registers as the non-local data paths do not affect the circuit timing directly. For consistency and completeness of the generated timing schedule, the data arrival times at the input registers are defined and the following constraints are included in the LP formulation for each input register R_l:

$$A_l = d_l - \left(D^l_{CQm} \text{ or } D^l_{DQm}\right) \;\; \forall R_l : |Fan - in(R_l)| = 0. \tag{4.16}$$

Note that (4.16) is computed only for input registers.

4.4 LINEARIZATION OF THE TIMING ANALYSIS

The *non-linear* max and min functions in the constraints shown in (4.3), (4.4), (4.5) and (4.6) present a major challenge in solving the clock skew scheduling problem. A method is introduced in this chapter in order to replace the non-linear constraints with linear constraints. Although theoretically inequivalent, it is demonstrated that the same results are obtained with the original non-linear programming (NLP) model and the novel linear programming (LP) model problems in experimentation with ISCAS'89 benchmark circuits.

The proposed linearization method is described in Section 4.4.1. The LP model for the clock period minimization problem of non-zero clock skew, level-sensitive circuits is offered in Section 4.4.2.

4.4.1 Modified Big M (MBM) Method

The linearization of the constraints which exhibit non-linear behavior is a commonly applied procedure in operations research [92]. When possible, non-linear constraints are manipulated to derive linear constraints, which are inherently easier to solve. In this work, a collection of linearization procedures is applied to the non-linear constraints of the timing analysis problem. The collection of these procedures is called the *Modified big M (MBM) method*. It is considered reasonable to denominate the collection of linearization procedures the *MBM method*, as the research is developed by an inspiration from the "big M method" [92]. The big M method is a special case of the simplex algorithm [92] which has applications in a completely distinct set of problems with respect to the MBM method. The only similarity between the big M method and the MBM method is the use of the constant M in both methods. The constant M symbolically represents a *very large* positive number used to assign an overwhelmingly large penalty to a variable in the objective function in order to increase the priority of the variable in the optimization process.

The collection of linearization procedures composing the MBM method is presented in Table 1. For a minimization type LP problem—subject to constraints that have min and max functions—the transformations listed in Table 1 are applied to replace *non-linear*

constraints with linear constraints. Note that only relevant constraints and relevant terms of the objective function are included in Table 1.

Define a finite set N, consisting of the variables $N = \{a, b, c, \ldots, n\}$. Consider all variables in the finite set N to be elements of the real numbers set $N = \{a, b, c, \ldots, n\} \subset \Re$. The objective function Z is a linear function of the variables $\{a, b, c, \ldots, n\}$ and is defined $Z : \Re^{|N|} \rightarrow \Re$. There are no limitations on variables being inter-dependent, provided the linearity of the constraints is preserved.

Two different linearization scenarios are presented in Table 1. In the first scenario [linearization of $a = \max(b, c)$ expression], the variable a is constrained to be the greater of the variables b and c. The constraint is replaced with two new constraints, explicitly requiring the variable a to be greater than or equal to the variables b and c. The initial constraint and the relaxed constraints are equivalent if either of the following conditions holds:

1. Equality condition is observed for at least one of the inequalities, while the other inequality operation returns true,
2. Equality condition is observed for both inequalities.

The cost function denoted by the product Ma is *added* to the objective function. The product Ma is overwhelmingly large with respect to other cost functions in the objective function as a result of the highly-weighed cost figure (recall the very large coefficient M).

Table 1: Modified *Big M* transformations.

$\min\ Z$	$\rightarrow$	$\min\ (Z + Ma)$
$a = \max(b, c)$	$\rightarrow$	$a \geq b$
		$a \geq c$
$\min Z$	$\rightarrow$	$\min\ (Z - Ma)$
$a = \min(b, c)$	$\rightarrow$	$a \leq b$
		$a \leq c$

Thus, Ma is given the highest priority in the minimization process. As a result, the greater of the variables b and c is assigned to variable a.

The relaxation method in the second scenario [linearization of $a = \min(b, c)$ expression] is also presented in Table 1. In this case, the cost function Ma is *subtracted* from the objective function in order to exploit the maximum value to be assigned to the variable a.

Similar to its implementation in the big M method, the constant M is defined *sufficiently* large, but as small as possible. The selection of a value for the constant M depends on the solution space of a specific problem (problem constraints) and the objective function Z. Typically, the number M must be chosen significantly larger than the values of any parameter in the problem. However selection of an *extremely* large M may cause the LP solver to fail drastically [19]. A value of $M = 5000$ was experimentally found to be sufficiently large for the analysis of circuits with arrival and departure times up to 100 (time units), number of registers up to 2000, and number of data paths up to 30000. The interpretation of value assignment and the derivation of a lower bound on the constant M fall outside the scope of this work and will not be discussed. The comparison of the results between the non-linear problem and the MBM method-transformed linear problem is a straight-forward post-solution check.

4.4.2 Linear Programming (LP) Model

An LP model of the clock period minimization problem is generated through the application of the MBM method. There are five sets of constraints in the LP model. These sets are the latching [(4.1) and (4.2)], synchronization [(4.3) and (4.4)], propagation [(4.5) and (4.6)], validity [(4.14) and (4.15)] and initialization [(4.16)] constraints. Note that, for simplicity, the skew constraints [(4.7), (4.8) and (4.9)] are not included in the LP model for the clock period minimization problem. The finalized LP model for the clock period minimization problem is shown in Table 2.

The latching, validity and initialization constraints exhibit linear behavior. Therefore, these constraints remain unchanged in both the LP and NLP models as shown in constraints (*i-ii , vii-ix*) of the formulation. The synchronization constraints, however, are formed by

Table 2: LP model clock skew scheduling problem of level-sensitive circuits.

LP Model
min $T + M[\sum_{\forall R_j}(d_j + D_j) + \sum_{\forall R_k:\|Fan-in(R_k)\|\geq 1}(A_k - a_k)]$
subject to
(i) $a_f \geq H_f$
[Latching-Hold time]
(ii) $A_f \leq T - S_f$
[Latching-Setup time]
(iii) $d_i \geq a_i + D^i_{DQm}$
$d_i \geq T - C^L_W + D^i_{CQm}$
[Synchronization-Earliest time]
(iv) $D_i \geq A_i + D^i_{DQM}$
$D_i \geq T - C^L_W + D^i_{CQM}$
[Synchronization-Latest time]
(v) $a_f \leq d_{i_1} + D^{i_1 f}_{Pm} + T_{skew}(i_1, f) - T$
$\vdots$
$a_f \leq d_{i_n} + D^{i_n f}_{Pm} + T_{skew}(i_n, f) - T$
[Propagation-Earliest time]
(vi) $A_f \geq D_{i_1} + D^{i_1 f}_{PM} + T_{skew}(i_1, f) - T$
$\vdots$
$A_f \geq D_{i_n} + D^{i_n f}_{PM} + T_{skew}(i_n, f) - T$
[Propagation-Latest time]
(vii) $A_f \geq a_f$
[Validity-Arrival time]
$(viii)$ $D_f \geq d_f$
[Validity-Departure time]
(ix) $A_l = d_l - (D^l_{CQm}$ or $D^l_{DQm})$, $\forall R_l : \|Fan - in(R_l)\| = 0$
[Initialization]

the max function and exhibit non-linear behavior. The MBM method is used on the synchronization constraints in order to generate linear constraints for the LP model problem (constraints *iii* and *iv*). For instance, (*iii*) depicts the replacement of the non-linear constraint presented in (4.3) with two linear constraints, where d_i is greater than or equal to both operands of the max function, $\left(a_i + D^i_{DQm}\right)$ and $\left(T - C^L_W + D^i_{CQm}\right)$. Note that the cost function Md_i is added to the objective function. Propagation constraint on the latest data arrival time (4.6), exhibits similar non-linearity with the synchronization constraints such that the max function is used. The linearized propagation constraints in the LP model are shown in (*vi*). In the LP model, the variable A_f is greater than or equal to the expressions $\left(\left[D_i + D^{if}_{PM} + T_{skew}(i,f)\right] - T\right)$, evaluated for each fan-in path of register R_f. In the formulation, fan-in paths of R_f are indexed by the parameter n.

Unlike other non-linear constraints in the formulation, the propagation constraint on the earliest arrival time a_f is modeled by the min function. In this type of linearization, a_f is set to be less than or equal to each operand of the min function. As shown in (*v*), the expressions $\left[d_i + D^{if}_{Pm} + T_{skew}(i,f) - T\right]$ evaluated for each fan-in path of register R_f are included in the finalized LP model.

In order to illustrate the derivation of the NLP and LP model formulations for a clock period minimization problem, a simple synchronous circuit is investigated. The NLP model formulation of the clock period minimization problem is presented in Appendix A. The non-linear constraints in the NLP problem formulation are linearized using the MBM method described in Section 4.4.1. The finalized LP model formulation for the clock period minimization problem is presented in Appendix B.

4.5 AN EXAMPLE AND EXPERIMENTAL RESULTS

The circuit network shown in Figure 16 is analyzed in order to illustrate the application of the proposed linearization procedure. Without affecting the generality of the solution, zero setup and hold times and zero internal delays are considered ($S_i = H_i = D_{CQ} = D_{DQ} = 0$). A single phase synchronization scheme with 50% duty cycle is selected as shown in Figure 17.

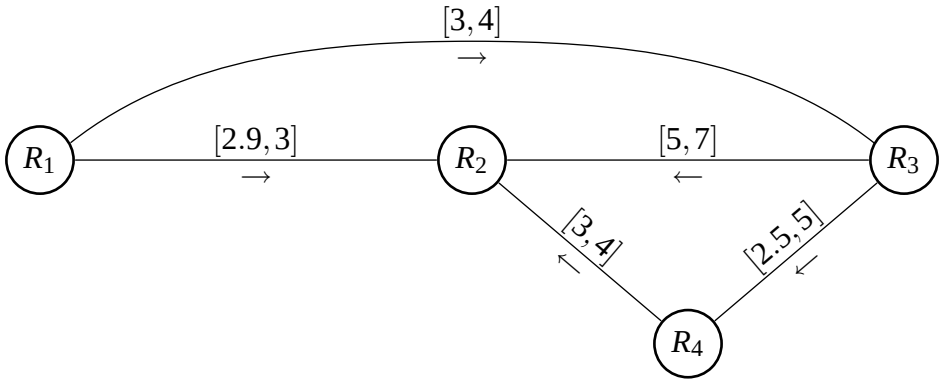


Figure 16: A simple synchronous circuit.

Given single-phase synchronization under zero and non-zero clock skew operation, the clock period minimization problems of three different synchronous circuits with same circuit topology are formulated. These circuits are:

1. Zero clock skew, edge-sensitive circuit,
2. Zero clock skew, level-sensitive circuit,
3. Non-zero clock skew, level-sensitive circuit.

The simpler (in terms of timing analysis) circuit, which is used as the basis of comparison for other circuits, is the *zero clock skew, edge-sensitive* circuit. The minimum clock period of a zero clock skew, edge-sensitive circuit is defined by the maximum data propagation time in the circuit [43]. Thus, the synchronous circuit network presented in Figure 16 has a minimum clock period of $T = D_{PM}^{32} = 7$ (time units) when used with edge-triggered flip-flops.

The second synchronous circuit of interest is the *zero clock skew, level-sensitive* circuit. In order to design a level-sensitive synchronous circuit, each flip-flop in the given circuit topology is replaced with a level-sensitive latch. Zero clock skew, level-sensitive circuits exhibit improved circuit performance due to time borrowing.

Clock skew scheduling is applied to the zero clock skew, level-sensitive circuit to generate the *non-zero clock skew, level-sensitive circuit*. This circuit exhibits performance improvement due to the simultaneous consideration of time borrowing and clock skew scheduling.

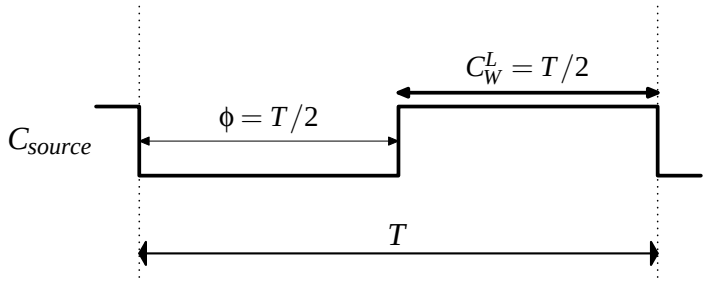
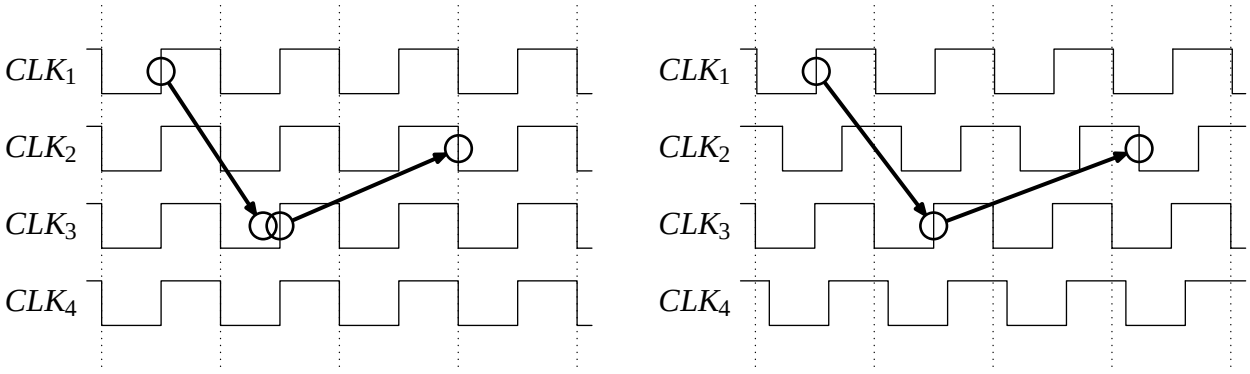


Figure 17: A single-phase synchronization clock with a 50% duty cycle.

The generic LP model shown in Table 2 (page 40) is used in problem formulation. The commercial optimization package CPLEX (v7.5) [36] is used to solve for these clock period minimization problems of the generated synchronous circuits. In experiments, the primal and dual simplex optimizers of CPLEX are used. The worst case analysis shows that the simplex method and its variants may require exponential number of steps to reach an optimal solution [19]. However, a vast amount of practice has confirmed that in most cases, the number of iterations to reach an optimal solution is polynomial [19].

Note that the number of problem constraints m is proportional to the number of registers r and the number of local data paths p in the circuit. Let s denote the number of input registers for which the initialization constraints are defined. In the LP model clock period minimization problem shown in Table 2, there are eight (8) constraints for each register, two (2) constraints for each local data path, and one (1) constraint for each input register. Thus, the number of constraints in the problem formulation is $m = 8r + 2p + s$. The minimum clock period T is a problem variable. Also, there are five (5) problem variables defined for each register leading to a total number of $n = 5r + 1$ variables in the problem formulation. The exact computational complexity cannot be determined since the internal presolver, matrix-sparsity checker and large-scale optimizer [19, 36] routines employed within CPLEX are proprietary and unknown.

In the analysis, the minimum clock period for the zero clock skew, level-sensitive circuit is calculated as 4.66 (time units), which is a 33% improvement over the zero clock skew,



Zero Skew	Non-Zero Skew	Critical Path
$A_3 = 1.66 = D_1 + 4 - 4.66$	$A_3 = 2.025 = D_1 + 4 + (0.05 - 0) - 4.05$	$R_1 \rightarrow R_3$
$A_2 = 4.66 = D_3 + 7 - 4.66$	$A_2 = 4.05 = D_3 + 7 + (0 - 0.925) - 4.05$	$R_3 \rightarrow R_2$

Figure 18: Zero and non-zero clock skew timing schedules for the circuit in Figure 16.

edge-sensitive synchronous circuit. Note that the percentage improvement is calculated by the expression $100(T_{old} - T_{new})/T_{old}$. As stated earlier, clock skew scheduling is applied to the level-sensitive circuit in order to generate the non-zero clock skew, level-sensitive circuit. The calculated minimum clock period of 4.05 for the non-zero clock skew, level-sensitive circuit is a 13% improvement over the zero clock skew, level-sensitive circuit and a 42% improvement over the zero clock skew, edge-sensitive circuit. Note that 13% improvement is only due to clock skew scheduling, while 42% improvement is due to time borrowing *and* clock skew scheduling. Further analysis of the time borrowing and clock skew scheduling effects on circuit timing are presented in Section 4.5.2. The clocking schedules and the data propagation on the critical paths of the circuit in Figure 16 are shown in Figure 18. In Figure 18, the clocking schedule for the zero clock skew circuit is shown on the left, with a minimum clock period of $T = 4.66$. Non-zero clock skew scheduling results with a minimum clock period of $T = 4.05$ is shown on the right. For non-zero clock skew scheduling, the optimal clock signal delays at the register are $t_1 = 0.05$, $t_2 = 0.925$, $t_3 = 0$ and $t_4 = 0.475$. The arrows represent data signal propagation on the respective critical paths. Note that

unlike the case presented in Figure 18, the critical paths for zero and non-zero clock skew scheduling need not be identical.

4.5.1 Level-Sensitive Synchronous Circuit State of Operation

Presence of data path loops (cycles) and transient state errors are two major issues that need to be identified in the timing analysis of level-sensitive circuits. As discussed in Section 4.2, the iterative algorithm offered in [64] suffers from excessive run times and produces false negative outputs in presence of data path loops [73]. In [73], modifications are offered for the iterative algorithm in order to detect and handle the effects of data path loops in the circuit. Also in [73], it has been shown that synchronous circuits are prone to suffer from transient state errors. The transient state errors occur due to the non-unique solution sets of the problem parameters, discussed (within a different context) in Section 4.3.2. In circuits under transient state errors, setup violations occur in certain registers after the system is initiated from a reset state. The arrival and departure times may not be stable at start-up, in which case these times change during initial clock cycles, constituting the transient state. As circuit operation progresses in time, the arrival and departure times converge to their steady-state values.

There are two major conventions in evaluating the transient errors and determining the steady-state behavior. The first convention overlooks the transient errors and presumes that the departure times converge to the opening edge of the driving clock, which is the expected schedule for the steady-state of operation. The second convention is more strict in that transient state errors are not permitted. The first convention is adopted more commonly within the proposed solution algorithms and leads to a generally acceptable solution unless the transient state operation of the level-sensitive circuit is decisive to overall circuit operation. Given that the second convention is adopted, the reset state is preferably extended until the steady state of operation is reached [73].

The LP model proposed in this work assumes the transient-state operation of a level-sensitive circuit to be negligible. The aim of the generated model is to solve for the steady-state timing scheduling problem. The simplex algorithm-based LP solver directs the gradual

advancement of parameter values as they are enforced by the LP model (Table 2). Previously offered algorithms are vulnerable to potential fallacies caused by data path loops due to their iterative nature. However, in the presented procedure, complications posed by the presence of data path loops are resolved within the mechanics of the LP solver without significantly affecting the run time or quality of the solution. If the problem remains feasible, the timing parameters for the steady state operation of the circuit are calculated.

In order to illustrate the described phenomenon, the steady-state optimal timing schedule for the ISCAS'89 benchmark circuit `s27` is presented in Figure 19. Simplifications of $D^{if}_{Pm} = D^{if}_{PM}, \forall R_i \rightarrow R_f$ and $S_i = H_i = D^i_{CQ} = D^i_{DQ} = 0$ are considered. The circuit `s27` has one input register and a data path loop consisting of two other registers. The data signal departs from input register R_3 and perpetually propagates on the loop between R_1 and R_2. The minimum clock period is calculated to be 4.1, where the pre-computed data propagation times are indicated on the circuit graph.

In Figure 19, the data propagations occurring on all data paths of the $s27$ benchmark circuit are analyzed. As defined in Section 2.3, the subscripts to the clock signal indicate the register being synchronized by the clock signal. The clock signals most likely are not aligned in time due to the non-identical clock delays to their respective destination registers. The clock signal C_3 at the input register R_3 has no delay in time with respect to the clock signal at the clock source ($t_3 = 0$). Hence, the origin of the clock signal at the source is aligned with the origin of C_3. The clock signals C_1 and C_2 however, are shifted in time by $t_1 = 3.8$ and $t_2 = 1.3$ relative to the origin of the clock signal at the source. The horizontal axis of Figure 19 represents the time, where the beginning $(k-1)T$ of the k-th clock cycle of C_3, is defined as the local time reference, with an assigned value of zero. In Figure 19, the numbers associated with the leading (enabling) and trailing (latching) edges of the clock signals label the times with respect to the local time reference. The arrows illustrate the propagation between the registers and are drawn to scale. Illustration of the data propagation on three consecutive clock cycles are sufficient to analyze the behavior of the data path loop of the benchmark circuit $s27$. Arbitrary cycles labeled the k-th, $(k+1)$-th and $(k+2)$-th clock cycles are selected. The solid arrows represent the data propagation during the selected clock cycles. For instance, the propagation between R_3 and R_1 is represented by the arrows initiating from

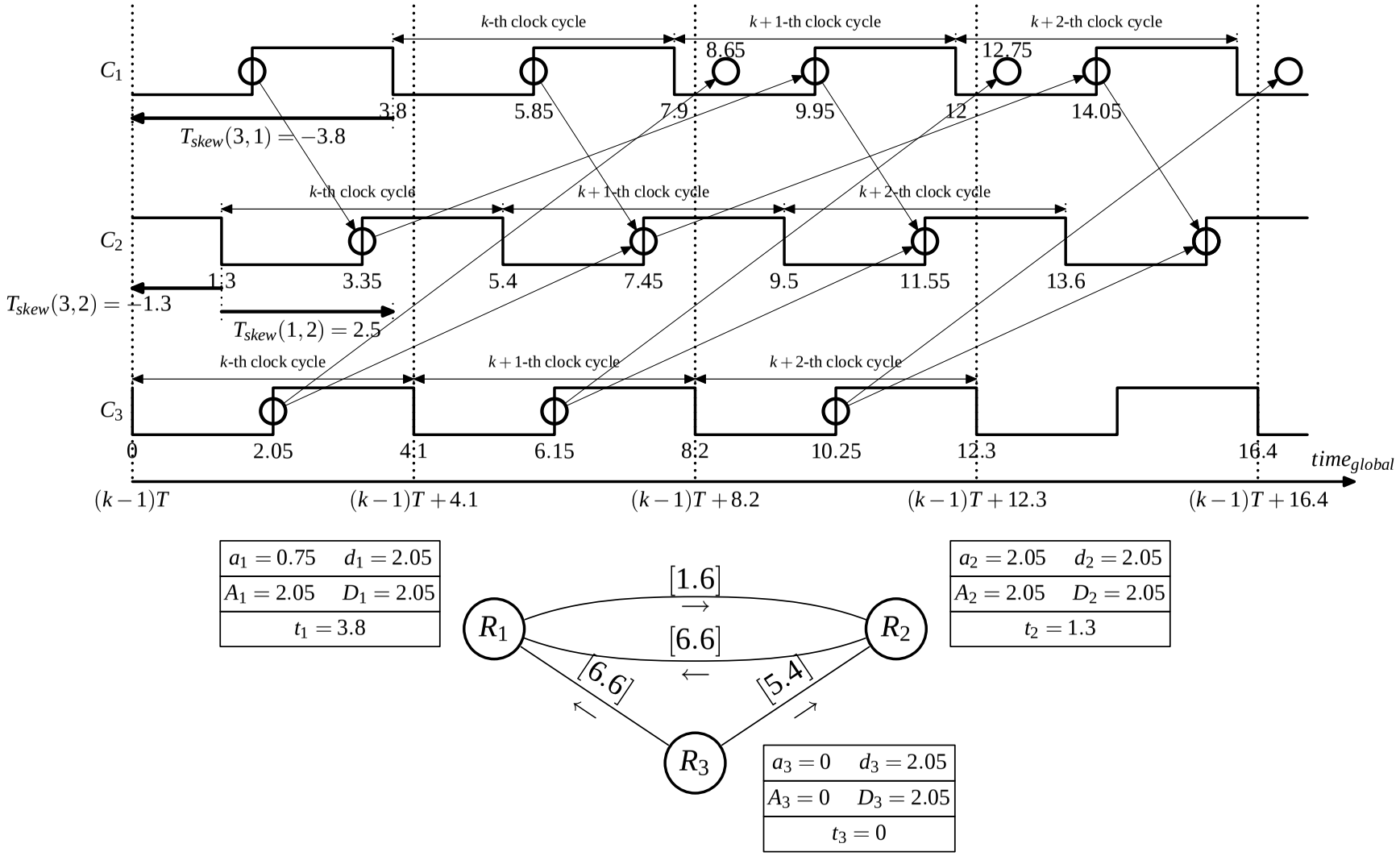


Figure 19: The optimized timing schedule for `s27` operable with T = 4.1.

the C_3 row at times 2.05 and 6.15, and concluding at the C_2 row at times 8.65 and 12.75, respectively. Data propagation on the data path loop between the registers R_1 and R_2 is visible by the cross-structured arrows initiating and concluding in the corresponding clock signal rows. Note that the calculated nominal arrival and departure times are illustrated on the circuit graph, inside the boxes associated with each node.

In steady-state of operation, the departure times of the registers that constitute a data path loop converge to the beginning of their respective clock cycles. The circuit `s27` in Figure 19 is analyzed in order to provide a better insight on how the latest departure times converge to a certain value in the steady-state. Define a variable ϵ, where ϵ is a very small period of time. Suppose that a deviation of ϵ occurs in the departure time of the data signal from R_3. The signal departure from R_3 occurs at time $2.05+\epsilon$, delaying the arrival times at R_1 and R_2 by ϵ. The departure from R_2 is gradually delayed by ϵ every turn, which in turn delays the arrival time at R_1. The arrival and departure times cumulatively increase in each turn of the data signal around the loop. Eventually, the signal arrivals at the latches occur during the non-transparent state of the latches. At this point, the signal departure times return to their starting values, which are the trailing edges of their respective clock cycles. It is evident that the arrival times will finally be restored to their initial values when the source of the deviation vanishes. Thus, the assignment of the time-varying departure times to the leading edges of the synchronizing clock signals is referred to as the steady-state of operation for the synchronous circuit.

4.5.2 Experimental Results on ISCAS'89 Benchmark Circuits

The timing analysis method described in this chapter is applied to the selected suite of ISCAS'89 benchmark circuits in order to derive the performance results and illustrate the efficiency and accuracy of the presented method. The original ISCAS'89 benchmark circuits are edge-sensitive synchronous circuits without any timing information. The timing information for the benchmark circuits is generated explicitly with an algorithm, where the type, size and fan-out of a gate are included in the computed combinational gate delay.

Level-sensitive implementations of the ISCAS'89 benchmark circuits are generated by replacing each flip-flop in the original benchmark circuit with a level-sensitive latch as discussed in Section 4.5. In experimentation, a single phase clock signal with a duty cycle of 50% is selected. Without affecting the generality of the solution, the setup and hold times and the internal delays are assumed to be zero ($S_i = H_i = D_{CQ} = D_{DQ} = 0$). The consideration of these numeric constants in an actual problem is straightforward. *Edge-sensitive* and *level-sensitive* synchronous circuit implementations are analyzed for *zero* and *non-zero clock skew scheduling* applications. The effects of time borrowing and clock skew scheduling in circuit implementation are investigated. The results of the analyses—computed on a 440MHz Sun Ultra-10 Workstation—are presented in Table 3. For each circuit, the following data are listed—the circuit name, the number of registers r and the number of paths p, the clock periods T_{FF}^{noskew} for a zero skew circuit with flip-flops, T_{L}^{noskew} for a zero skew circuit with latches, T_{FF}^{skewed} for a non-zero skew circuit with flip-flops, T_{L}^{skewed} for a non-zero skew circuit with latches, and T_{L}^{r} for a non-zero skew circuit where the clock delays to I/O registers are restricted to be equal. The subscripts FF, L represent circuit topologies for flip-flop based and latch-based circuits, respectively. The superscripts $noskew, skewed$ indicate zero or non-zero clock skew scheduling. Also listed are the calculation time of T_{L}^{skewed}, t_{L}^{skewed}, and the clock period improvements I_{L}^{TB}, I_{FF}^{CSS} and I_{L}^{TBCSS}, where the superscripts $TB, CSS, TBCSS$ stand for time borrowing, clock skew scheduling and both, respectively.

The minimum clock periods calculated for the edge-sensitive synchronous circuits under zero and non-zero clock skew scheduling (T_{FF}^{noskew} and T_{FF}^{skewed}, respectively) are borrowed from [43]. It is reported in [43] that, due to clock skew scheduling, an average improvement of 30% is reported in the minimum clock period for the ISCAS'89 benchmark circuits.

The experimental results shown in Table 3 present significant improvements in the minimum clock period for synchronous circuits with level-sensitive latches. In digital synchronous circuits, utilizing latches as storage elements instead of flip-flops may result in up to 33% improvements of the minimum clock period under zero clock skew (for single-phase, 50% duty cycle clock synchronization). On the ISCAS'89 benchmark circuits, an average of 15% improvement is observed when the flip-flops are replaced by latches (under zero clock skew). This level of improvement is solely due to time borrowing.

Table 3: Clock skew scheduling results for level-sensitive ISCAS'89 benchmark circuits.

Circuit Info			**Zero CS**		**I (%)**	**Non-Zero CS**		**I (%)**			**t (sec)**	**R**	**I (%)**
Circuit	r	p	T_{FF}^{noskew}	T_{L}^{noskew}	I_{L}^{TB}	T_{FF}^{skewed}	T_{L}^{skewed}	I_{FF}^{CSS}	I_{L}^{TBCSS}	I_{L}^{CSS}	t_{L}^{skewed}	T_{L}^{r}	I_{L}^{r}
s27	3	4	6.6	5.4	18	4.1	4.1	38	38	24	0.02	4.1	38
s208.1	8	28	12.4	8.6	31	4.9	5.2	60	58	40	0.01	7.6	39
s298	14	54	13.0	10.6	18	9.4	9.4	28	28	11	0.02	10.6	18
s344	15	68	27.0	18.4	32	18.4	18.4	32	32	0	0.03	18.4	32
s349	15	68	27.0	18.4	32	18.4	18.4	32	32	0	0.03	18.4	32
s382	21	113	14.2	10.3	27	8.5	8.5	40	40	17	0.04	8.7	39
s386	6	15	17.8	17.3	3	17.3	17.3	3	3	0	0.03	17.3	3
s400	21	113	14.2	10.4	27	8.6	8.6	39	39	17	0.05	8.8	38
s420.1	16	120	16.4	12.6	23	6.8	7.2	59	56	43	0.04	10.3	37
s444	16	113	16.8	12.4	26	9.9	9.9	41	41	20	0.07	9.9	41
s510	6	15	16.8	14.8	12	14.8	14.3	12	15	3	0.02	14.8	12
s526	21	117	13.0	10.6	18	9.4	9.4	28	28	11	0.05	10.6	18
s526n	21	117	13.0	10.6	18	9.4	9.4	28	28	11	0.05	10.6	18
s641	19	81	83.6	66.2	21	61.9	61.9	26	26	6	0.05	63.1	25
s713	19	81	89.2	71.2	20	63.8	63.8	28	28	10	0.05	65.0	27
s820	5	10	18.6	18.3	2	18.3	18.3	2	2	0	0.01	18.3	2
s832	5	10	19.0	18.8	1	18.8	18.8	1	1	0	0.01	18.8	1
s838.1	32	496	24.4	20.6	16	8.3	9.1	66	63	56	0.28	15.6	36
s938	32	496	24.4	20.6	16	8.3	9.1	66	63	56	0.31	15.6	36
s953	29	135	23.2	21.2	9	18.3	18.3	21	21	14	0.10	21.2	9
s967	29	135	20.6	17.9	13	16.2	16.6	21	19	7	0.08	17.9	13
s991	19	51	96.4	91.6	5	79.4	79.4	18	18	13	0.02	79.4	18
s1196	18	20	20.8	16.0	23	10.8	7.8	48	63	51	0.03	16.0	23
s1238	18	20	20.8	16.0	23	10.8	7.8	48	63	51	0.01	16.0	23
s1423	74	1471	92.2	86.4	6	77.4	75.8	16	18	12	1.10	75.8	18
s1488	6	15	32.2	29.0	10	29.0	29.0	10	10	0	0.02	29.0	10
s1494	6	15	32.8	29.6	10	29.6	29.6	10	10	0	0.01	29.6	10
s1512	57	415	39.6	34.8	12	34.8	34.8	12	12	0	0.28	34.8	12
s3271	116	789	40.3	29.8	26	28.6	28.6	29	29	4	0.69	29.0	28
s3330	132	514	34.8	23.4	33	17.8	17.8	49	49	24	0.49	23.2	33
s3384	183	1759	85.2	77.4	9	67.4	67.4	21	21	13	1.88	76.2	11
s4863	104	620	81.2	75.4	7	69.0	69.0	15	15	8	0.64	69.0	15
s5378	179	1147	28.4	23.2	18	22.0	22.0	23	23	5	1.66	22.0	23
s6669	239	2138	128.6	124.6	3	109.8	109.8	15	15	12	3.62	109.8	15
s9234	228	247	75.8	64.8	15	54.2	54.2	28	28	16	4.59	59.2	22
s9234.1	211	2342	75.8	64.8	15	54.2	54.2	28	28	16	3.88	59.2	22
s13207	669	3068	85.6	67.4	21	57.1	57.1	33	33	15	14.86	57.1	33
s15850	597	14257	116.0	92.8	20	83.6	83.6	28	28	10	76.96	83.6	28
s15850.1	534	10830	81.2	71.4	12	57.4	57.4	29	29	20	58.89	57.4	29
s35932	1728	4187	34.2	34.1	0	20.4	20.4	40	40	40	80.03	20.4	40
s38417	1636	28082	69.0	54.8	21	42.2	42.2	39	39	23	603.49	43.0	39
s38584	1452	15545	94.2	76.4	19	65.2	65.2	31	31	16	321.74	64.8	31
Average					15			30	27	14			24

Utilizing non-zero clock skew, an even higher improvement is possible. Improvements up to 63%—over flip-flop based synchronous circuit with zero clock skew—are observed. The average improvement in the minimum clock period for ISCAS'89 benchmark circuits is 27%. This level of improvement is due to simultaneous application of clock skew scheduling and consideration of time borrowing.

In the 27% simultaneous improvement for non-zero clock skew, level-sensitive circuits, the improvement due to time borrowing is 15% and the improvement due to clock skew scheduling is 14%. It is interesting to note that the improvements achieved through time borrowing and clock skew scheduling are not additive. Time borrowing and clock skew scheduling target the same resource in performance improvement, the slack propagation time on local data paths. There is a limited amount of slack propagation time on the critical paths and a circuit where time borrowing is abundantly realized, cannot benefit as much from clock skew scheduling. It has been shown however, that even though time borrowing and clock skew scheduling are battling effects (battling for the same resource), dramatically shorter clock periods are achievable through the collaboration of both effects.

The zero clock skew, level-sensitive circuit implementation is analogous to the circuits targeted in previous research presented in [8, 74]. Note that unlike the unit-delay-per-gate approach used in [8, 74], the combinational logic delays are calculated by assuming different delay times for each logic gate type and considering effects of fanout on the propagation time. Thus, the obtained results are not *directly* comparable to the previously published algorithm results. However, presuming the accuracy and correctness of both procedures, the results listed for T_L^{noskew} can be considered as the results that are calculated by the methods presented in [8, 74].

Simultaneous consideration of time borrowing and clock skew scheduling in the proposed procedure results in higher improvements (I_L^{TBCSS}) compared to consideration of time borrowing and constant clock skew (T_L^{noskew}). Therefore, the procedure presented in this work is superior to the methods presented in [8, 74] in terms of performance improvement.

4.6 OPTIMALITY OF THE LP FORMULATION

The operational constraints (latching [(4.1) and (4.2)], synchronization [(4.3) and (4.4)] and propagation [(4.5) and (4.6)] constraints) accurately model the timing of level-sensitive synchronous circuits. However, the synchronization and propagation constraints are non-linear, leading to a non-linear programming (NLP) problem formulation. Remember from Section 4.4 that the NLP formulation is illustrated for a simple circuit in Appendix A.

Typical NLP problems, especially for large-scale systems, are very hard to solve efficiently. Consequently, alternative modeling and solution procedures to solve for the timing constraints of level-sensitive circuits are of interest for researchers. As discussed in Section 4.4, a novel linearization procedure that generates an LP formulation is presented in this dissertation. Neither the iterative solution methods proposed in [8, 74] nor the LP model problem presented in this work are equivalent to the original non-linear problem. These alternative solution methods are proposed in order to generate results that are as close as possible to the optimal solution in relatively shorter run times.

In this section, a Mixed-Integer (Linear) Programming (MIP) [19, 92] formulation that is equivalent to the NLP formulation of the clock skew scheduling problem for level-sensitive circuits is described. A MIP problem is a linear programming problem in which some or all of the problem variables are constrained to be integers [19, 92]. If the integer variables are further constrained to take only 0 or 1 values, these variables are called *binary* variables.

In general, a MIP problem can be solved optimally (granted enough time) or within a close proximity of the optimal solution [19]. A typical MIP problem, although generally harder to solve than an LP problem of similar size, is generally easier to solve than an NLP problem of similar size [92]. In experimentation, the MIP problems generated for the clock skew scheduling problem of level-sensitive ISCAS'89 benchmark circuits are solved optimally.

In order to generate the MIP formulation for the clock skew scheduling problem of level-sensitive circuits, the non-linear synchronization and propagation constraints in Table 2 (page 40) are re-modeled using binary variables. Remember from Section 4.4.1 that the non-linearity of the synchronization and propagation constraints are due to the *max* and *min* functions. The transformations in Table 4 can be used to model a constraint with a max

Table 4: MIP modeling of a constraint with a max or a min function.

$y_i = \max(x_i, x_j, \ldots, x_k)$	$y_i = \min(x_i, x_j, \ldots, x_k)$
$y_i \geq x_i$	$y_i \leq x_i$
$y_i \geq x_j$	$y_i \leq x_j$
$\vdots$	$\vdots$
$y_i \geq x_k$	$y_i \leq x_k$
$y_i + (Bx_i - 1)M \leq x_i$	$y_i + (1 - Bx_i)M \geq x_i$
$y_i + (Bx_j - 1)M \leq x_j$	$y_i + (1 - Bx_j)M \geq x_j$
$\vdots$	$\vdots$
$y_i + (Bx_k - 1)M \leq x_k$	$y_i + (1 - Bx_k)M \geq x_k$
$Bx_i + Bx_j + \cdots + Bx_k \geq 1$	$Bx_i + Bx_j + \cdots + Bx_k \geq 1$
$Bx_i, Bx_j, \ldots, Bx_k$ binary	$Bx_i, Bx_j, \ldots, Bx_k$ binary

function or a min function using a binary variable. In Table 4, y_i, x_i, x_j and x_k are continuous variables. A binary variable Bx_a is defined for each operand x_a ($x_a \in \{x_i, x_j, \ldots, x_k\}$) of the max or min function. For operand x_i of the max function shown on the left hand side of Table 4, for instance, the binary variable Bx_i is defined. The parameter M is a *sufficiently* large constant, similar to its definition in Section 4.4.1.

For a non-linear constraint with the max function in the form $[y_i = \max(x_i, x_j, \ldots, x_k)]$, y_i is constrained to be greater than or equal to each one of the operands. For the max function to hold, equality condition must be true for at least one of these inequalities (multiple equalities occur when two or more identical operands are the maximal value). Binary variables are used in order to enforce the equality of at least one of these inequalities. The assignment of 0 or 1 to the binary variables Bx_a either constrain y_i to be less than or equal to x_a or

constrain y_i to be strictly greater than x_a. In particular for operand x_i, when $Bx_i = 1$, the relevant constraints become the following:

$$y_i \geq x_i \tag{4.17}$$

$$y_i \leq x_i \tag{4.18}$$

which simplifies to the equality $y_i = x_i$ through x_i being the largest of the operands $x_i, x_j, \ldots, x_k$. On the other hand, if $Bx_i = 0$, the relevant constraints become the following:

$$y_i \geq x_i \tag{4.19}$$

$$y_i - M \leq x_i \tag{4.20}$$

which simplifies to $y_i > x_i$. The transformation for a non-linear constraint with the min function in the form $[y_i = \min(x_i, x_j, \ldots, x_k)]$ is similar, as shown on the right hand side of Table 4.

Using the transformation procedures defined in Table 4 on the non-linear synchronization and propagation constraints, the MIP problem is constructed for the clock skew scheduling problem of level-sensitive circuits. The constraints from Table 2 that change in the MIP problem are shown in Table 5.

The MIP formulations of the clock skew scheduling problem are derived for the ISCAS'89 benchmark circuits. The MIP formulations are equivalent to the NLP formulation, thus, the results of the MIP problems are optimal for each benchmark circuit. These MIP problems are solved in order to observe the potential deviations from optimality because of modeling the NLP problem as an LP problem as described in Section 4.4.1. It is observed that *all* of the ISCAS'89 suite of benchmark circuits are solved optimally with the LP model problem.

For small-sized circuits, the MIP formulation can be preferred due to its guarantee for optimality. However, as the number of registers and paths grow, the solutions of the MIP problems can suffer from very long run times (can be practically insolvable). In order to compare the run times of the MIP problems with the run times of the LP problems, experiments are performed on the ISCAS'89 benchmark circuits. Note that, the run times for the LP problems are reported in Table 3 under column t_L^{skewed}.

Table 5: MIP model clock skew scheduling problem of level-sensitive circuits.

MIP Model
min T
subject to
(iii) $d_i \geq a_i + D^i_{DQm}$
$d_i \geq T - C^L_W + D^i_{CQm}$
$d_i + (Ba_i - 1)M \leq a_i + D^i_{DQm}$
$d_i + (BTa_i - 1)M \leq T - C^L_W + D^i_{CQm}$
[Synchronization-Earliest time]
(iv) $D_i \geq A_i + D^i_{DQM}$
$D_i \geq T - C^L_W + D^i_{CQM}$
$D_i + (BA_i - 1)M \leq A_i + D^i_{DQM}$
$D_i + (BTA_i - 1)M \leq T - C^L_W + D^i_{CQM}$
[Synchronization-Latest time]
(v) $a_f \leq d_{i_1} + D^{i_1 f}_{Pm} + T_{skew}(i_1, f) - T$
$\vdots$
$a_f \leq d_{i_n} + D^{i_n f}_{Pm} + T_{skew}(i_n, f) - T$
$a_f + (1 - Bd_{i_1} f)M \geq d_{i_1} + D^{i_1 f}_{Pm} + T_{skew}(i_1, f) - T$
$\vdots$
$a_f + (1 - Bd_{i_n} f)M \geq d_{i_n} + D^{i_n f}_{Pm} + T_{skew}(i_n, f) - T$
[Propagation-Earliest time]
(vi) $A_f \geq D_{i_1} + D^{i_1 f}_{PM} + T_{skew}(i_1, f) - T$
$\vdots$
$A_f \geq D_{i_n} + D^{i_n f}_{PM} + T_{skew}(i_n, f) - T$
$A_f + (BD_{i_1} f - 1)M \leq D_{i_1} + D^{i_1 f}_{PM} + T_{skew}(i_1, f) - T$
$\vdots$
$A_f + (BD_{i_n} f - 1)M \leq D_{i_n} + D^{i_n f}_{PM} + T_{skew}(i_n, f) - T$
[Propagation-Latest time]

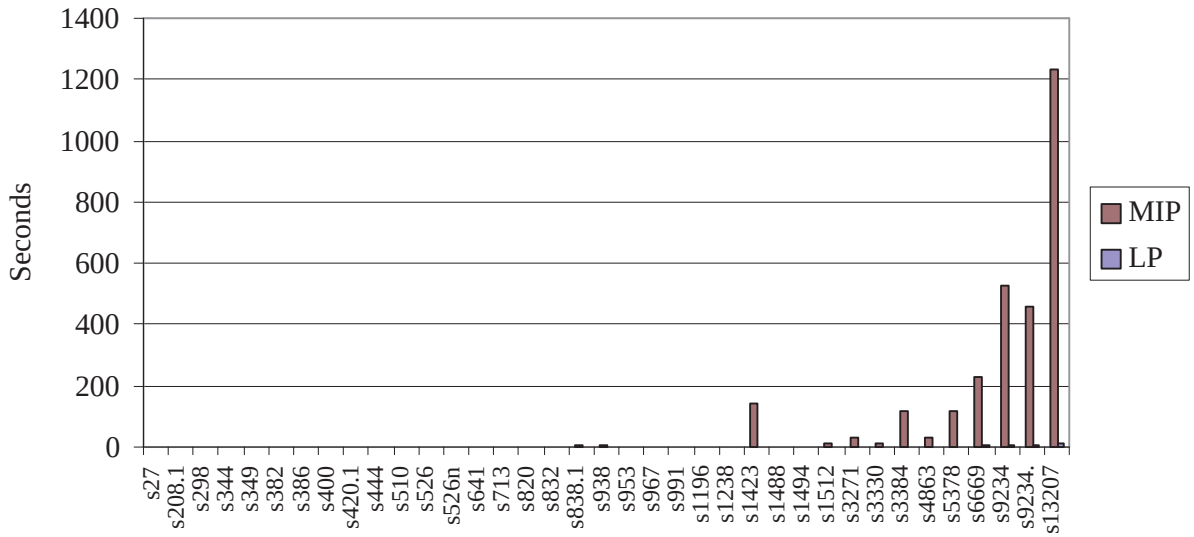


Figure 20: Run times under 1500 seconds for the LP and MIP formulations.

In Figure 20, the ISCAS'89 benchmark circuits whose run times are below 1500 seconds using CPLEX (v7.5) simplex solver on a 440MHz Sun Ultra-10 Workstation are shown. For smaller circuits, both LP and MIP run times are below a few seconds, thus cannot be visualized with the scale used in Figure 20. For `s1423` and larger benchmark circuits, whose number of paths exceed a thousand, a significant gap between the run times of the LP and MIP problems is observed. For larger circuits, the MIP run times can get extremely worse compared to the LP run times. For instance, the MIP problem run time for `s38417` is 286496 seconds, while the LP problem run time is only 603 seconds.

The run time experiment results shown in Figure 20 demonstrate the advantages of using the LP formulation versus the MIP formulation. It is demonstrated that the LP formulation suggests a scalable alternative to the accurate MIP model. It is expected that the run times for industry-size integrated circuit will benefit even more from the simplifications of the LP formulation. The results of the LP formulation for the ISCAS'89 benchmark circuits are empirically shown to be equal to the optimal results. These empirical results do not

guarantee the optimality of results for *all* circuits using the LP formulation. However, these results suggest the general accuracy of the LP formulation for the clock skew scheduling problem of level-sensitive circuits in leading to optimal or close to optimal results.

4.7 VERIFICATION AND INTERPRETATION OF RESULTS

Some edge-sensitive synchronous circuits are inoperable with level-sensitive latches as briefly stated in Section 2.1. For such circuits, the clock skew scheduling problem is infeasible. The presented timing analysis procedure detects the infeasibility of a such problem and provides diagnostics messages. The slack and excess values associated with each constraint can be examined in the sensitivity analysis output provided by an LP solver. Even though the details will not be discussed here, careful interpretation of the sensitivity output leads to the identification of the necessary modifications on the circuit topology to achieve the desired operating frequency. The sensitivity analysis output of the LP solver CPLEX for the timing analysis discussed in Appendices A and B is presented in Appendix C.

The interpretation of the timing schedule for a synchronous circuit presents a model to investigate the effects of zero and non-zero clock skew scheduling on synchronous circuit operation. In the rest of this section, the timing schedules generated for the synchronization of the ISCAS'89 benchmark circuit `s938` with zero and non-zero clock skew scheduling are analyzed. The analyses include the data distributions for various parameters, which are presented in Section 4.7.1. The verification of clock skew values is discussed in Section 4.7.2. In Section 4.7.2, the skew constraints of Section 4.1.4 are used to derive lower and upper bounds on clock skew.

4.7.1 Parameter Data Distributions

In Section 3.2, data propagation time D_P^{if} is defined as the period of time the data is processed in the combinational logic block of a local data path $R_i \rightarrow R_f$. Without loss of generality, an empirical calculation method is used to calculate the data propagation times of each local

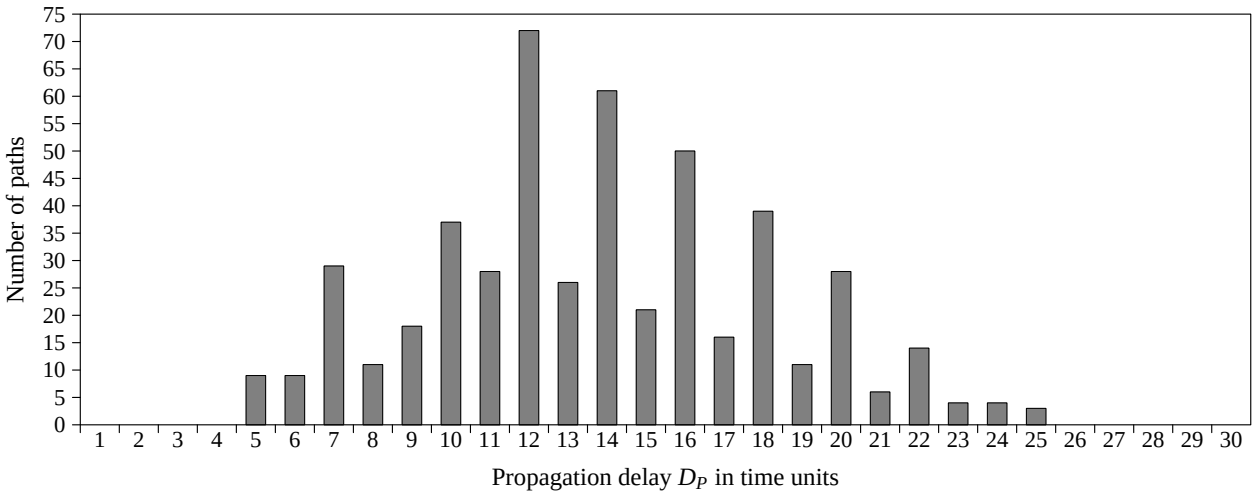


Figure 21: Data propagation times for `s938` with 32 registers and 496 data paths.

data path of a circuit (a simple fan-out delay model is used as timing data is not included in the ISCAS'89 benchmark circuits). The distribution of the calculated data propagation times for the ISCAS'89 benchmark circuit $s938$ is illustrated in Figure 21. In this figure, the height of each bar corresponds to the number of paths within a given delay range. For example, there are nine (9) paths with delays between 4 and 5 time units.

Effective path delay [43] is defined as the time period between the departure of the data signal from the initial register and the arrival of the same data signal at the final register. The effective path delay of a local data path differs from data propagation delay, because of the additional propagation time provided by clock skew and the time borrowing property of level-sensitive synchronous circuits. Note that in level-sensitive synchronous circuits, the effective path delay is defined within a permissible range instead of a fixed value, as the arrival and departure times are indeterminate. The nominal effective path delay is determined when the arrival and departure times are realized in run-time as certain values in the permissible ranges $[a_f, A_f]$ and $[d_i, D_i]$, respectively. Specifically, the shortest effective path delay occurs when the data signal departs at its latest time D_i from the initial register R_i and arrives at its earliest arrival time a_f at the final register R_f. The longest effective path delay is realized

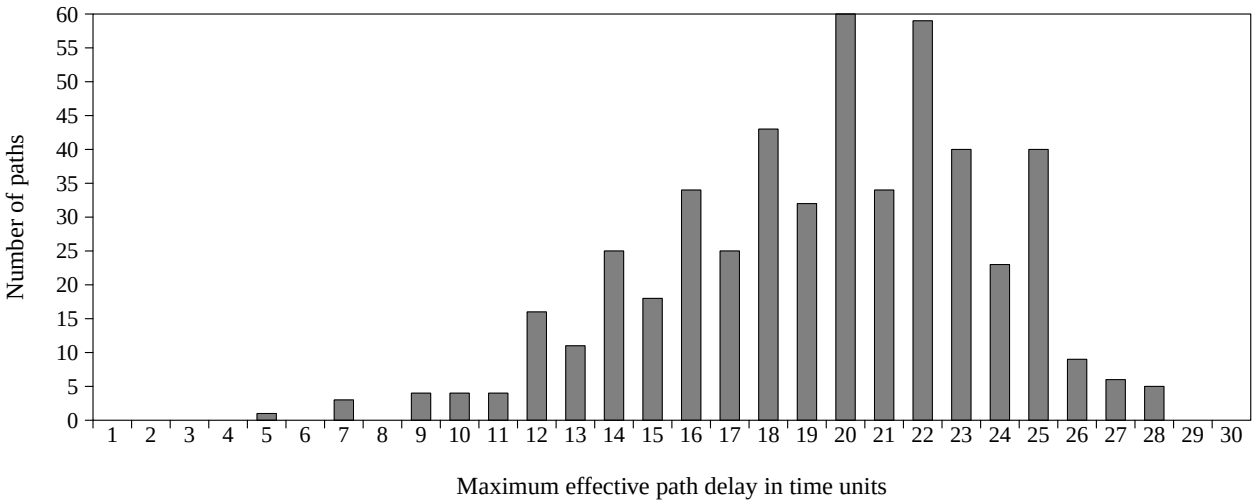


Figure 22: Maximum effective path delays in data paths of `s938` for zero clock skew.

by the earliest departure d_i of the data signal from R_i and latest arrival A_f at R_f. Hence, the interval for the effective path delay of level-sensitive synchronous circuits can be defined as:

$$a_f - D_i - T_{skew}(i,f) + T \leq \text{Effective path delay} \leq A_f - d_i - T_{skew}(i,f) + T. \qquad (4.21)$$

In this work, the longest effective path delay is investigated in order to illustrate the effects of clock skew and time borrowing on data propagation. The aim is to observe the increase in the effective path delay of a circuit, which in turn leads to a higher operating frequency, by replacement of flip-flops with latches and introducing non-zero clock skew. Observe that the distribution of the propagation delays for the $s938$ benchmark circuit presented in Figure 21 is exactly the same as the distribution of the effective path delay of the same benchmark circuit $s938$, when operational with flip-flops (under zero clock skew). In circuits with flip-flops, the effective path delays are determinate $\left[D_P^{if} - T_{skew}(i,f)\right]$ as the data departures occur at the active transition of the clock signal.

The distribution of the maximum effective path delays of the level-sensitive `s938` circuit with zero clock skew scheduling is shown in Figure 22. Note that the maximum effective path

delay is calculated by the expression $[A_f - d_i - T_{skew}(i,f) + T]$. The target clock period is $T = 20.6$. The height of each bar corresponds to the number of paths with an effective path delay within a given range. It is observed by comparing Figures 21 and 22 that the maximum effective path delays are *increased* in the level-sensitive circuit, as well as providing a smaller minimum clock period $\left(T_{FF}^{noskew} = 24.4 \text{ v.s. } T_L^{noskew} = 20.6\right)$. The increase in the effective path delays is due to time borrowing. Accumulation of effective path delay values slightly below or above the minimum operating clock period $T = 20.6$ is visible. Note that the effective path delay having larger values than the minimum clock period is a sufficient but not a necessary condition for time borrowing. Thus, local data paths where the effective path delay is calculated to be smaller than $T = 20.6$ may still benefit from time borrowing. Furthermore, it can be observed that certain data paths in the circuit benefit more from time borrowing, realizing an effective path delay close to the theoretical limit of $\left[T + C_W^L - T_{skew}(i,f)\right]$.

4.7.2 Skew Analysis

As discussed throughout this chapter, non-zero clock skew scheduling in synchronous circuits permits smaller clock periods. Note that in presence of non-zero clock skew, the effective path delay for the data signal over a data path most likely gets *smaller* compared to its value observed in zero clock skew scheduling. This fact is directed by (4.21) (T gets smaller). However, as the minimum clock period T gets smaller, the percentage of the data paths, on which the effective path delay exceeds the minimum clock period, significantly increases (see Figure 23). The target clock period is $T = 9.09$. The height of each bar corresponds to the number of paths with an effective path delay within a given range. The effect of clock skew on improving the minimum clock period is visible by comparing the histograms presented in Figures 22 and 23.

The skew constraints [(4.7), (4.8) and (4.9)] introduced in Section 4.1.4 can be included in the LP model (Table 2) in order to ensure the correctness of the solution. The skew constraints not only constitute an extra measure to check for the feasibility of the solution but are also used in collecting statistical data on clock skew values. Interpretation of (4.11)

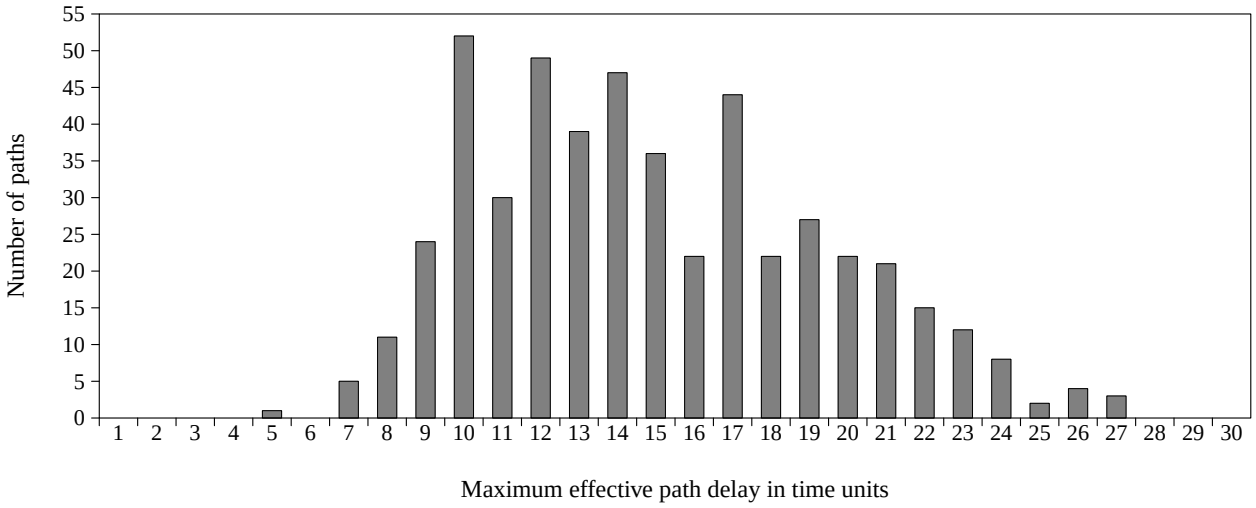


Figure 23: Maximum effective path delays for `s938` for non-zero clock skew.

and (4.13) lead to the upper and lower bound definitions for the clock skew. In order to generate an expression for the upper bound, (4.11) rewritten as:

$$D_i + D_{PM}^{if} - T + T_{skew}(i, f) \leq T - S_f. \tag{4.22}$$

In (4.22), the earliest possible time is assigned to D_i in order to realize the upper bound on clock skew. The earliest possible time that a data signal departs from a latch is D_{CQ} later than the leading edge of the clock signal, $\left(T - C_W^L + D_{CQ}\right)$. Reordering the expression gives the upper bound on clock skew:

$$T_{skew}(i, f) \leq T + C_W^L - D_{PM}^{if} - D_{CQ} - S_f. \tag{4.23}$$

The lower bound on the clock skew is derived similarly from (4.13), which leads to:

$$a_f + D_{Pm}^{if} \geq T - T_{skew}(i, f) + H_f. \tag{4.24}$$

In order to derive the lower bound, the data arrival time at R_f must be considered to occur at its latest possible time. The latest data arrival time is the setup time S_f earlier than the trailing edge of the clock signal, $T - S_f$. Thus, the lower bound on the clock skew is:

$$T_{skew}(i, f) \geq T - T - D_{Pm}^{if} + S_f + H_f. \tag{4.25}$$

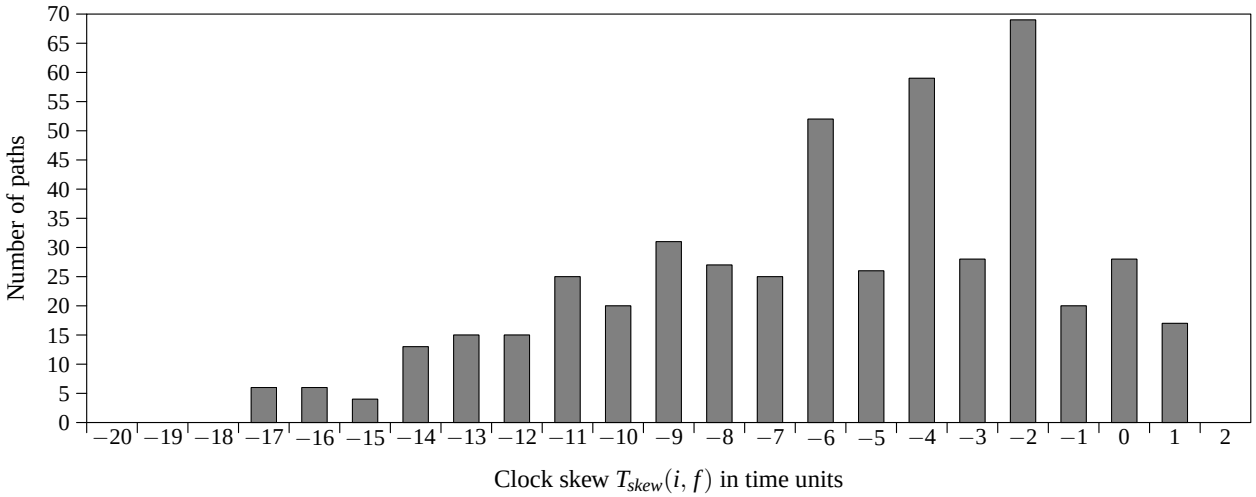


Figure 24: Distribution of the clock skew values of the non-zero clock skew case for `s938`.

Combining (4.23) and (4.25), the theoretical limits on clock skew is expressed as follows:

$$-D_{Pm}^{if} + S_f + H_f \leq T_{skew}(i, f) \leq T + C_W^L - D_{PM}^{if} - D_{CQ} - S_f. \tag{4.26}$$

Recall that in experimentation, the parameters D_{DQ}, D_{CQ}, S_f, H_f are considered zero and 50% duty cycle is selected for the single-phase synchronization clock signal. In order to evaluate the upper and lower bounds on clock skew in this simplified case, the parameters are substituted in (4.26):

$$-D_{Pm}^{if} \leq T_{skew}(i, f) \leq 1.5T - D_{PM}^{if}. \tag{4.27}$$

Specifically on the ISCAS'89 benchmark circuit `s938`, the clock skew bounds are verified using the experimental values shown in Figure 21. For the benchmark circuit `s938` with a minimum clock period of 9.09, the minimum and maximum propagation delays are calculated to be 5 and 24.4, respectively. Thus, the value set for the clock skew variable on the data paths of `s938` is constrained by $-24.4 \leq T_{skew}(i, f) \leq 8.64$.

The distribution of the clock skew values of `s938`, when operable with a minimum clock period of 9.09, is presented in Figure 24. The target clock period is $T = 9.09$. The height of

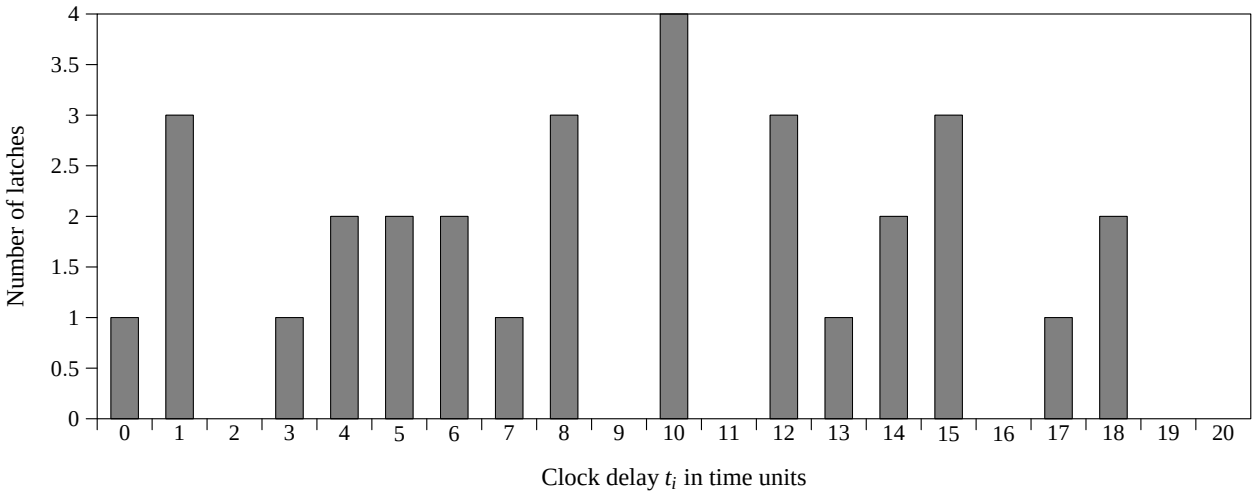


Figure 25: Distribution of the clock delay values of the non-zero clock skew case for `s938`.

each bar corresponds to the number of paths formed by sequentially adjacent pair of registers which have a clock skew within the given range. The calculated clock skew values are within the derived limits, most of which are negative. Negative clock skew between registers help improve the minimum clock period of the synchronous circuit due to the additional time it provides for data signal propagation. The data paths, on which positive skew is recorded, most likely occur due to two reasons. The first reason is the presence of data path loops within the circuit. The second reason are the—faster—paths which provide extra time for neighboring critical paths.

The distribution of the clock delays to each register presented in Figure 25. The target clock period is $T = 9.09$. The height of each bar corresponds to the number of latches being driven by a clock signal with a time delay within the given range.The distribution is significantly wide-spread, ranging from 0 to 19 (time units), where the minimum clock period is $T = 9.09$. If the clock tree network of the synchronous circuit is implemented to accommodate for these nominal clock delays, operation at the target minimum clock period is achieved.

4.8 FURTHER CONSIDERATIONS

The presented LP model formulation is demonstrated to be effective for the static timing analysis of synchronous circuits. In the analyses, classical circuit implementations are investigated, such that, no additional timing dependencies between the registers of a synchronous circuit, other than the dependencies leading to the predefined timing constraints, are prescribed. As application-specific integrated circuit (ASIC) design becomes wide-spread, the need for a timing analysis model, which can accommodate for application-specific constraints, becomes essential. Unlike the previous work in [8, 74], the presented LP formulation is highly amenable to such modifications, constituting a well-defined timing analysis framework.

The following describes a potential problem in the timing analysis of SOC designs. In an ASIC/SOC implementation, the clock signal distribution between different IP blocks (or clock domains) are subject to consideration as well as the distribution of the clock signal within an IP block. The clock delays to the *I/O* registers of a synchronous IP block are less flexible compared to the clock delays to the *internal* registers. It is likely that the timing analysis will be performed on individual IP blocks by the vendors, without a priori information of the application environment. Therefore, timing violations may occur on outgoing (non-local, intra-block) data paths, as the clock skew on these data paths will be unaccounted for in the initial computation. A simple solution to avoid timing violations between the IP blocks is to equalize all I/O register clock delay values. This situation is illustrated on a sample IP block in Figure 26. In the presented framework, additional timing constraints enforcing the equality of the clock signal delays can easily be integrated into the constraints set of the LP model problem presented in Table 2. Such modification is applied to the formulation and the resulting problem is experimented on the ISCAS'89 benchmark circuits. The results are presented in Table 3, under the columns T_L^r and I_L^r for the minimum clock period and improvement in minimum clock period (compared to zero clock skew, edge-sensitive circuit), respectively. The restriction of the clock delays of the I/O registers leads to 24% improvement on average, down from the 27% average improvement for the unrestricted case.

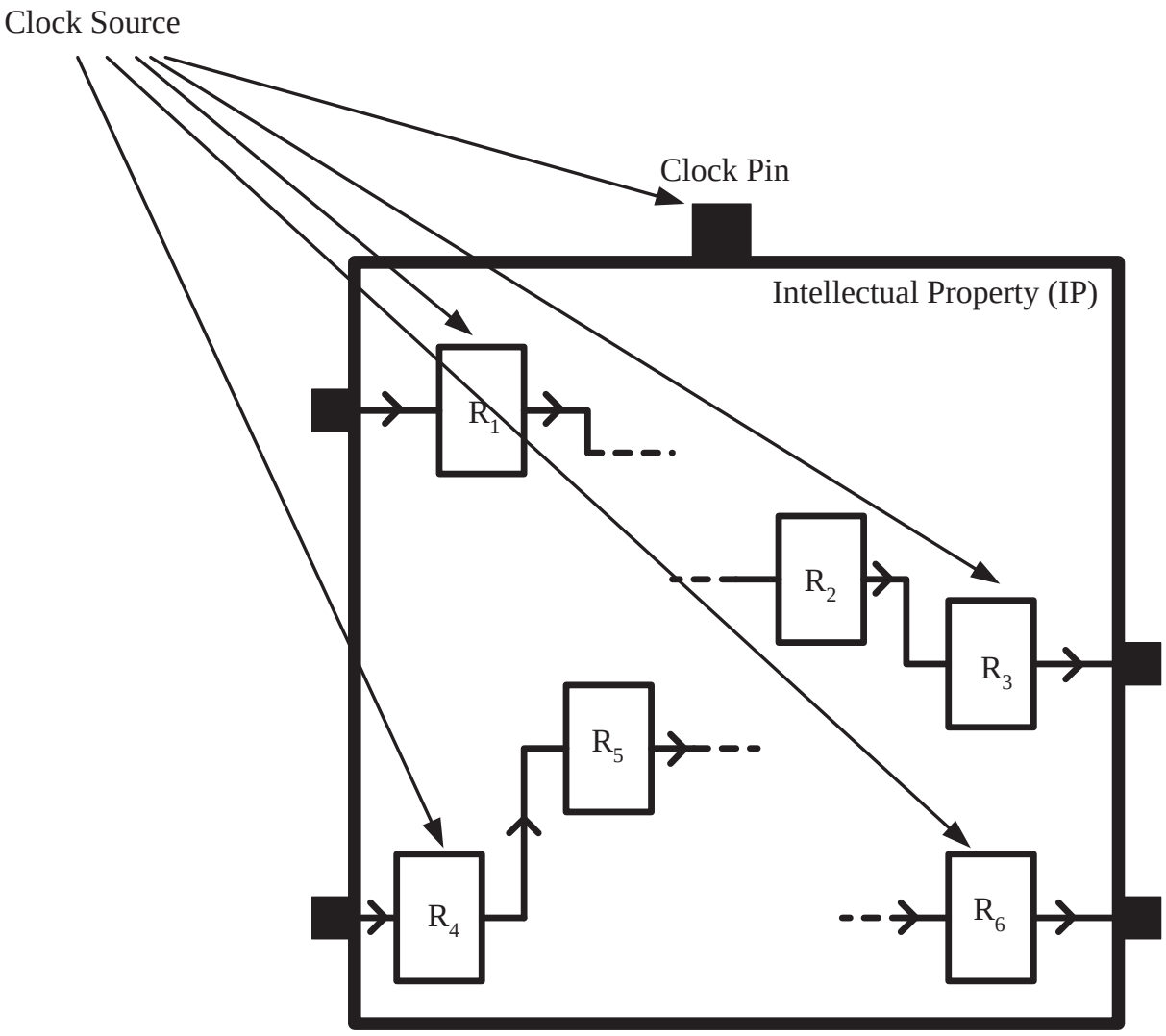


Figure 26: Additional timing requirements of an IP block.

Another commonly encountered design constraint is to implement a predetermined—possibly non-optimal—clock tree network for the synchronous circuit. If the clock tree topology is predetermined, the minimum clock period problem must be solved with known clock delays to each register. The LP model presented in Table 2 can easily be modified to account for such changes, by assigning the given clock signal delays to the respective clock delay variables.

4.9 SUMMARY

The timing analysis and optimization of synchronous circuits are subject to non-zero clock skew (intentional or not) and other effects of process parameter variations. In this chapter, novel design and timing analysis procedures are presented, where the application of clock skew scheduling is considered simultaneously with time borrowing. The simultaneous application is used to improve the performance of level-sensitive synchronous circuits in permitting shorter clock periods. The described procedure is the first to integrate non-zero clock skew scheduling in an automated fashion into the design and analysis level-sensitive circuits. The procedure is based on a stand-alone LP model formulation (to be solved by any standard LP solver) which constitutes a generic automated framework for the design and analysis of level-sensitive synchronous circuits.

The optimality of the results generated by the novel LP model is empirically confirmed against the optimal results of a precise MIP model. The LP model formulation is shown to be highly generic and amenable to accommodate application-specific timing constraints. Experiments on ISCAS'89 benchmark circuits demonstrate improvements of 27% shorter clock periods on average.

The work presented in this chapter has been published in [75, 77, 78, 81]. In [75, 77, 78] and this chapter, the synchronization of level-sensitive circuits with a single-phase clock is considered due to its popularity and the simplicity of accompanying timing analysis. The proposed formulation (Section 4.3) and solution (Section 4.4) procedures can be modified such that these procedures are applied to multi-phase synchronized circuits. Enhancements on the formulation of the timing analysis of synchronous circuits for multi-phase synchronization are discussed in Chapter 6.

5.0 CLOCK SKEW SCHEDULING WITH DELAY INSERTION

In mainstream digital integrated circuit design flow, delay insertion is used as a post-processing step in order to solve the short-path (hold time) timing violations of synchronous circuits [65]. The drawbacks of delay insertion, such as increased circuit area and power dissipation, are usually disregarded in favor of achieving a feasible timing schedule.

In this chapter, a delay insertion algorithm that improves the efficiency and results of clock skew scheduling is presented. By systematic delay insertion performed simultaneously with clock skew scheduling, a higher operating speed or improved reliability is achieved. It is known that the minimum clock period of a synchronous circuit achievable through clock skew scheduling is limited by the uncertainties of the data propagation times on local data paths [21] and the total data propagation times on data path loops [56]. It is shown for the first time in this work that the reconvergent local data paths introduce an additional theoretical limit on the minimum clock period of a synchronous circuit achievable through clock skew scheduling. This limitation caused by reconvergent paths is theoretically derived and a novel delay insertion method is defined in order to mitigate this limitation. In the rest of this chapter, it is assumed that reconvergent paths are the *dominant limiting factor* on the minimum clock period of a synchronous circuit achievable through clock skew scheduling over other limiting factors of delay uncertainty and data path cycles. This assumption does not invalidate the generality of the work, it is adopted in order to simply the presentation of the described limitation.

The limitation on the minimum clock period caused by reconvergent paths is derived for both edge-triggered and level-sensitive circuit implementations. It is shown that through systematic delay insertion, the limitation on the minimum clock period achievable through clock skew scheduling can be mitigated. For a scalable, fully-automated application, the

proposed delay insertion method is implemented as a Linear Programming (LP) problem. A topological analysis of a circuit (to identify the reconvergent paths) is not necessary in the presented LP problem, as the problem constraints are generated on the individual local data paths. The LP problem models the traditional clock period minimization problem of synchronous circuits through clock skew scheduling simultaneously with calculating the optimal delay values to be inserted on each local data path.

Note that a delay insertion method targeting the improvement of the operating frequency of synchronous circuits has previously been offered in [68]. This method and its variants are used in mainstream digital circuit design, where the circuits are designed to meet the register setup time requirements, and the hold time requirements are satisfied post-analysis by inserting appropriate delays. The delay insertion method presented in the chapter has certain similarities with the method offered in [68], but is fundamentally different in the following aspects:

1. The study in [68] is proposed for systems where the clock skews are pre-computed (post-clock tree synthesis). This study is proposed for systems where the optimal clock skews need to be computed (pre-clock tree synthesis),
2. The study in [68] is offered to mitigate short path constraint violations on local data paths. This study is proposed to mitigate *both* short and long path constraint violations that can occur with clock skew scheduling,
3. The study in [68] is offered to fix the timing violations on each local data path, where the timing of one local data path is independent from its neighboring local data paths (due to pre-computed clock skew). In this study, clock skew scheduling is considered, which leads to the interdependence of the timing of adjacent local data paths (due to the computation of optimal clock skew). Consequently, the topological orientation of local data paths (such as reconvergent paths) are of importance for this study. On a larger scale, the study in [68] is a delay insertion method at the combinational block level, while this study is a delay insertion method at the sequential block level of design hierarchy.

This chapter is organized as follows. In Section 5.1, the clock skew scheduling algorithm used in this chapter for edge-triggered circuits is reviewed. In Section 5.2, the optimal clock

skew scheduling results generated by the clock skew scheduling algorithms are analyzed. In Section 5.3, the proposed delay insertion method to improve the minimum clock period is introduced. In Section 5.4, main practical concerns in the modeling and application of the delay insertion method are discussed. The experimental results for the application of the proposed method on benchmark circuits are presented in Section 5.5. The delay insertion method is summarized in Section 5.6.

5.1 CLOCK SKEW SCHEDULING METHODS

Two clock skew scheduling methods—one for edge-triggered and one for level-sensitive circuits—are examined in the context of the proposed delay insertion method. The selected clock skew scheduling method for edge-triggered circuits is shown in Table 6. This simple and effective linear programming (LP) problem for the clock skew scheduling of edge-triggered circuits is introduced in [21]. In the problem constraints, the *permissible range* [43] for the clock skew values on each local data path are implied. The objective of the algorithm is to minimize the clock period T of a circuit, given the set of permissible range constraints. For the clock skew scheduling of level-sensitive circuits, the method presented in Table 2 of Chapter 4 (on page 40) is selected.

Both clock skew scheduling methods presented in Table 2 and Table 6 are used as the frameworks of formulation for the proposed delay insertion method. The modifications to

Table 6: LP model clock skew scheduling problem of edge-triggered circuits.

LP Model [21]	
min	T
s.t.	$T_{skew}(i,f) \leq T - D^{if}_{PM} - D^{i}_{CQM}$
	$T_{skew}(i,f) \geq -D^{if}_{Pm} - D^{i}_{CQm} + H_f$

these two methods necessary to incorporate the delay insertion method are discussed in Section 5.3.

5.2 DELAY INSERTION METHOD

When clock skew scheduling is applied to a synchronous circuit, a set of optimal values that satisfy the objective function (clock period minimization is considered in this work) are assigned to the clock delays at each register. Certain data paths become critical timing paths because of the distribution of these optimal clock delays. In this section, the consequences of criticality to the short and long path constraints of a reconvergent path are analyzed. It is demonstrated that when the short and long path constraints of a reconvergent path are critical, the minimum clock period can be improved via delay insertion. Note that criticality of the constraints of a reconvergent path adheres to the preliminary assumption that the limitation caused by this reconvergent path system is dominant over other limitations. For circuits where limitations caused by other factors are dominant, improvement through delay insertion is not possible. In experimentation, such circuits are reported to be one of the two cases where the delay insertion method is inapplicable (e.g. delay insertion method is not beneficial).

A *reconvergent data path system* is defined by two or more series of local data paths (*reconvergent paths*) with a common source register and a common sink register. The source and sink registers are called the *divergent* register R_d and the *convergent* register R_c, respectively. Let $p^{d\{i_1 \ldots i_n\}c}$ define a *reconvergent path* starting from register R_d, continuing through the *intermediate* registers $\{R_{i_1}, \ldots, R_{i_n}\}$ and ending at register R_c. The number of intermediate registers $r^{d\{i_1 \ldots i_n\}c} = n$ is a non-negative integer number ($n \in Z^+ \cup \{0\}$) and the path is acyclic [$\forall i_n, i_m : R_d \neq R_{i_n}$, $R_{i_n} \neq R_{i_m}$, $R_d \neq R_c$ and $R_{i_n} \neq R_c$]. In the sample circuit modeled in Figure 4 (page 13), for instance, there are two reconvergent paths between v_1 and v_3, p^{123} and p^{13}, where the numbers of intermediate registers for the two reconvergent paths of this circuit are $r^{123} = 1$ and $r^{13} = 0$, respectively. The *path delay* $PD^{d\{i_1 \ldots i_n\}c}$ of a reconvergent path $p^{d\{i_1 \ldots i_n\}c}$ is defined as the total data propagation time between the

divergent and convergent registers R_d and R_c, respectively, over the intermediate registers $\{R_{i_1}, \ldots, R_{i_n}\}$. The minimum and maximum path delays of this reconvergent data path are given by $PD_m^{d\{i_1 \ldots i_n\}c}$ and $PD_M^{d\{i_1 \ldots i_n\}c}$, respectively. The *system delay* SD^{dc} of a reconvergent data path system between divergent and convergent registers R_d and R_c is defined by the conjuncture of all the (reconvergent) path delays between registers R_d and R_c. The maximum system delay SD_M^{dc} of this reconvergent data path system is defined by the largest of the maximum path delays between R_d and R_c. Similarly, the minimum system delay SD_m^{dc} is defined by the smallest of the minimum path delays between R_d and R_c. If there are k number of reconvergent paths between R_d and R_c, labeled $p_A, p_B, \ldots, p_K$, then:

$$SD_m^{dc} = \min\left(PD_m^{p_A}, PD_m^{p_B}, \ldots, PD_m^{p_K}\right), \tag{5.1}$$

$$SD_M^{dc} = \max\left(PD_M^{p_A}, PD_M^{p_B}, \ldots, PD_M^{p_K}\right). \tag{5.2}$$

5.2.1 Example of Reconvergence

A simple reconvergent data path system formed by two reconvergent local data paths sharing the divergent and convergent registers R_1 and R_2, respectively, is shown in Figure 27. Note that as a special case, subscripts a and b are used to identify the two reconvergent local data paths p^{12_a} and p^{12_b}. Registers R_1 and R_2 are the divergent and convergent registers, respectively. The two reconvergent paths p^{12_a} and p^{12_b} form the reconvergent data path

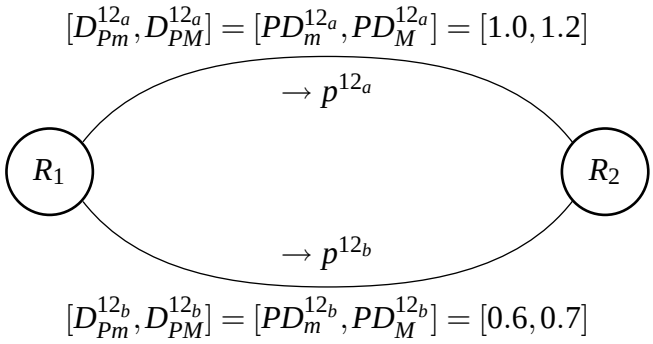


Figure 27: A simple reconvergent data path system.

system. For this simple reconvergent data path system, the path delay of each reconvergent path is the data propagation delay of the respective local data paths, $\left(PD_m^{12_a} = D_{P_m}^{12_a} = 1.0,\ PD_M^{12_a} = D_{PM}^{12_a} = 1.2\right)$ and $\left(PD_m^{12_b} = D_{P_m}^{12_b} = 0.6,\ PD_M^{12_b} = D_{PM}^{12_b} = 0.7\right)$. The minimum and maximum system delays are driven by the reconvergent data paths p^{12_b} and p^{12_a}, respectively:

$$SD_m^{12} = \min\left(PD_m^{12_a}, PD_m^{12_b}\right) = PD_m^{12_b} = 0.6, \tag{5.3}$$

$$SD_M^{12} = \max\left(PD_M^{12_a}, PD_M^{12_b}\right) = PD_M^{12_a} = 1.2. \tag{5.4}$$

Two circuits with the topology presented in Figure 27 are analyzed in Sections 5.2.2 and 5.2.3– the edge-triggered circuit S_{FF} and the level-sensitive circuit S_L, respectively.

5.2.2 Reconvergence in an Edge-Triggered Circuit

For edge triggered circuits, the data signals depart the registers clock-to-output delay (D_{CQ}) after the latching edge of the clock signal. Consequently in S_{FF}, the signal Q_1 (recall Figure 2 on page 8) departs R_1 clock-to-output delay D_{CQ} time after the positive clock edge and propagates along the reconvergent paths. In order to satisfy the short path constraints, the arrival of data signals X_{2_a} and X_{2_b} at R_2 must occur H_2 later than the positive edge of the previous clock cycle at R_2. Similarly, in order to satisfy the long path constraints, the arrivals must occur S_2 earlier that the positive edge of the current clock cycle at R_2:

$$H_2 \leq a_2 \leq A_2 \leq T - S_2. \tag{5.5}$$

Next, suppose clock skew scheduling for clock period minimization is applied to an arbitrary edge-triggered circuit which involves a reconvergent data path system. After clock skew scheduling, if at least one of the reconvergent paths becomes a critical timing path, the earliest and latest arrival times of the data signal at the critical convergent node are at marginal values. Accordingly for S_{FF}, the arrival times a_2 and A_2 satisfy

$$H_2 = a_2 \leq A_2 = T_{min} - S_2. \tag{5.6}$$

The constraints in (5.6) are illustrated in Figure 28. C_1 and C_2 are the clock signals syn-

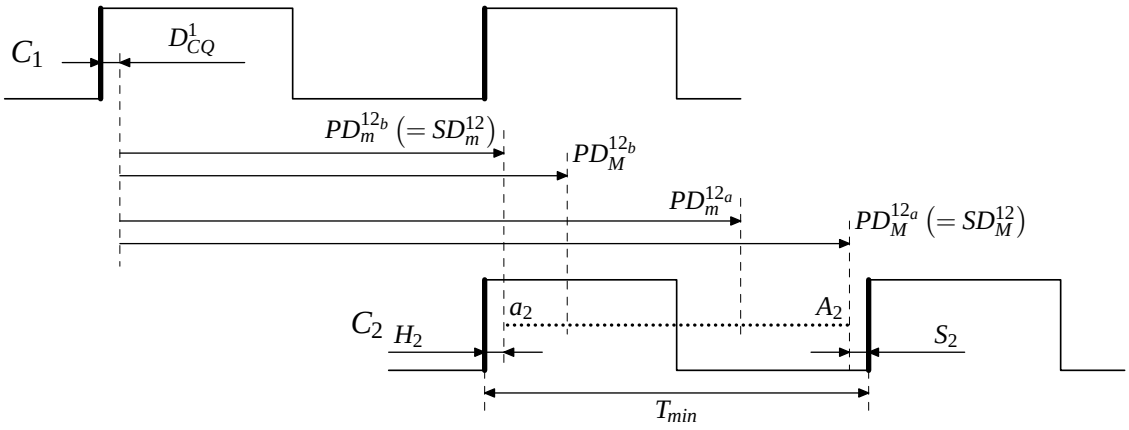


Figure 28: Timing of the edge-sensitive reconvergent system in Figure 27 after CSS.

chronizing registers R_1 and R_2, respectively. Also illustrated on Figure 28 is the separation between $A_2 + S_2$ and $a_2 - H_2$ defining the minimum clock period:

$$T_{min} = A_2 + S_2 - (a_2 - H_2). \tag{5.7}$$

Note that the data arrival times at R_2 are given by the propagation constraints (4.5), (4.6):

$$a_2 = \min\left(d_1 + D^{12_a}_{Pm} - T_{min}, d_1 + D^{12_b}_{Pm} - T_{min}\right) = d_1 + \min\left(D^{12_a}_{Pm}, D^{12_b}_{Pm}\right) - T_{min}, \tag{5.8}$$

$$A_2 = \max\left(D_1 + D^{12_a}_{PM} - T_{min}, D_1 + D^{12_b}_{PM} - T_{min}\right) = D_1 + \max\left(D^{12_a}_{PM}, D^{12_b}_{PM}\right) - T_{min} \tag{5.9}$$

Replacing the propagation constraints in (5.7) yields

$$T_{min} = D_1 + \max\left(D^{12_a}_{PM}, D^{12_b}_{PM}\right) - T_{min} + S_2 - \left[d_1 + \min\left(D^{12_a}_{Pm}, D^{12_b}_{Pm}\right) - T_{min} + H_2\right]. \tag{5.10}$$

Eq. (5.10) is simplified to

$$T_{min} = \max\left(PD^{12_a}_M, PD^{12_b}_M\right) - \min\left(PD^{12_a}_m, PD^{12_b}_m\right) + S_2 + H_2. \tag{5.11}$$

Following from (5.1) and (5.2), (5.11) is identical to

$$T_{min} = SD^{12}_M - SD^{12}_m + S_2 + H_2. \tag{5.12}$$

Substituting the numerical values and assuming zero internal register delays $D_{CQ} = D_{DQ} = S = H = 0$, the minimum clock period T_{min} of S_{FF} after clock skew scheduling is computed $T_{min} = 0.6$ time units.

Consider (5.12), showing the dependence of T_{min} on the algebraic difference between the maximum system delay and the minimum system delay between R_d and R_c (summed with the internal register delays S_f and H_f). The delay insertion method is proposed to modify these maximum and minimum system delays between R_d and R_c. The modification, when applicable, decreases the algebraic difference in (5.12). In S_{FF}, for instance, the minimum system delay between R_d and R_c is determined by $PD_m^{12_b}$ of path p^{12_b}. By inserting a delay element of 0.1 time units on p^{12_b}, the minimum and maximum path delays of this path are changed to $D_{Pm}^{12_b} = 0.7$ and $D_{PM}^{12_b} = 0.8$, respectively. More importantly, the minimum system delay between R_d and R_c is still determined by $PD_m^{12_b}$ of path p^{12_b}, which is now 0.7 instead of the original 0.6 time units. Both before and after delay insertion, the maximum system delay between R_d and R_c is determined by $PD_M^{12_a}$ of path p^{12_a}, which is a constant 1.2 time units. Therefore, the algebraic difference between the maximum and minimum system delays between R_d and R_c is improved from $(1.2 - 0.6 = 0.6)$ to $(1.2 - 0.7 = 0.5)$ time units. This delay insertion procedure for the circuit shown in Figure 27 is illustrated in Figure 29. The black circle in Figure 29 represents a delay element of [0.1,0.2] that is inserted on the reconvergent path p^{12_b}.

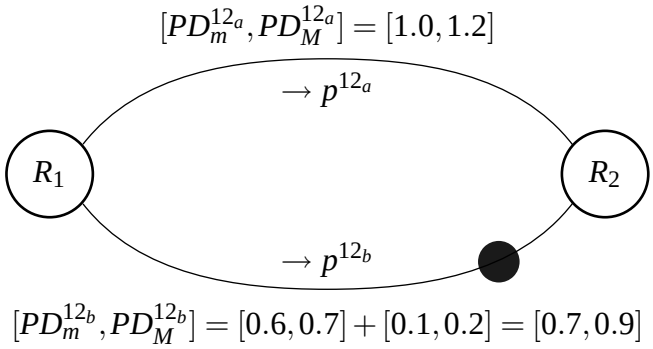


Figure 29: The simple reconvergent system in Figure 27 after delay insertion.

Note that for S_{FF}, inserting a delay element with a value in range [0.4,0.5] on p^{12_b} gives the *minimum possible* algebraic difference in (5.12), leading to the *minimum clock period obtainable through delay insertion* T^*_{min}. For S_{FF}, T^*_{min} evaluates to $T^*_{min} = 1.2 - 1.0 = 0.2$. It is shown that this minimum clock period obtainable through delay insertion depends on the maximum of the algebraic differences between the maximum and minimum path delays of each reconvergent path (after delay insertion).

Proposition: Let there be k number of reconvergent paths between R_d and R_c, labeled $p_A, p_B, \ldots, p_K$. The minimum possible algebraic difference between the maximum and minimum path delays of each reconvergent path between R_d and R_c *after delay insertion* is the minimum clock period T^*_{min} obtainable through delay insertion.

Let the minimum and maximum system delays define the real numbers interval Λ, such that:

$$\Lambda = [SD^{dc}_m, SD^{dc}_M] \tag{5.13}$$

By definition, the minimum *possible* algebraic difference between the maximum and minimum path delays of each reconvergent path *after delay insertion* (defining the minimum possible clock period) is the *minimum* length of interval Λ (after delay insertion).

In order to compute the minimum length $|\Lambda|$ of interval Λ achievable through delay insertion, the difference $[max(\Lambda) - min(\Lambda)]$ is computed. Recalling (5.1) and (5.2), the following is derived:

$$min(\Lambda) = SD^{dc}_m = \min\left(PD^{p_A}_m, PD^{p_B}_m, \ldots, PD^{p_K}_m\right), \tag{5.14}$$

$$max(\Lambda) = SD^{dc}_M = \max\left(PD^{p_A}_M, PD^{p_B}_M, \ldots, PD^{p_K}_M\right). \tag{5.15}$$

Let the real number delay intervals formed by the minimum and maximum delay values of the paths $p_A, p_B, \ldots, p_K$ be represented by $A, B, \ldots, K$, respectively. In other words, a delay interval L, associated with the path $p_L \in \{p_A, p_B, \ldots, p_K\}$ is formed by $L = [PD^{p_L}_m, PD^{p_L}_M]$. One of the following possibilities defining the expression $[|\Lambda| = max(\Lambda) - min(\Lambda)]$ must hold:

P1. A delay interval $M \in \{A, \ldots, K\}$ determines both the minimum $min(\Lambda)$ and maximum $max(\Lambda)$ values of the interval Λ. Then, $\Lambda = M$ and $|\Lambda| = |M| = max(\Lambda) - min(\Lambda) = max(M) - min(M)$,

P2. Otherwise, two non-identical delay intervals determine the minimum and maximum values of the interval Λ. Then, $\forall L \in \{A, \ldots, K\}$: $|\Lambda| = max(\Lambda) - min(\Lambda) > max(L) - min(L)$.

For systems satisfying (P1), the minimum length for Λ is already given by $|\Lambda| = |M|$. The minimum interval length, thus the minimum clock period, cannot be changed by delay insertion. For systems satisfying (P2), delay insertion method is used to modify one or more of the delay intervals in Λ in order to promote one of the delay intervals to become the interval M. In other words, systems satisfying (P2) are converted to systems satisfying (P1) through delay insertion. Delay insertion is performed into the logic network, thus, the systems delays and the interval Λ are modified with delay insertion. Note that both the minimum and maximum system delays can be modified with delay insertion. Therefore, it is not possible to predetermine which reconvergent path will be the determining path for the interval Λ after delay insertion.

In case (i) of Figure 30, a sample system satisfying (P1) is illustrated, where the delay interval D (associated with path p^D) determines the minimum length for Λ. No modification is necessary for such systems, as the minimum possible length for Λ is already observed.

In cases (ii) and (iii) of Figure 30, the application of the delay insertion method to a sample system satisfying (P2) is illustrated. Note that in case (ii), the minimum value in the Λ interval is determined identically by delay intervals C and D $[min(\Lambda) = PD_m^{p_C} = PD_m^{p_D}]$, while the maximum value is determined by delay interval B $[max(\Lambda) = PD_M^{p_B}]$. Delay insertion on a reconvergent path is similar to adding an offset to the interval, while preserving the interval length. If the optimal values of delay elements are inserted on each path, the minimum possible $|\Lambda|$ is achieved by asserting that the biggest delay interval $M \in \{A, \ldots, K\}$ becomes the interval Λ. In the modification of the sample system shown in cases (ii) and (iii) of Figure 30, the delay interval B is promoted to become this biggest delay interval M such that both $min(\Lambda)$ and $max(\Lambda)$ are determined by delay interval B (i.e. delay interval B becomes Λ). The intervals before and after delay insertion on the sample system are demonstrated in cases (ii) and (iii) of Figure 30, respectively.

There are two important points to note here. First, the solution set of the inserted delay values is not unique (remember similar discussions in Sections 4.3.2 and 4.5.1). For instance,

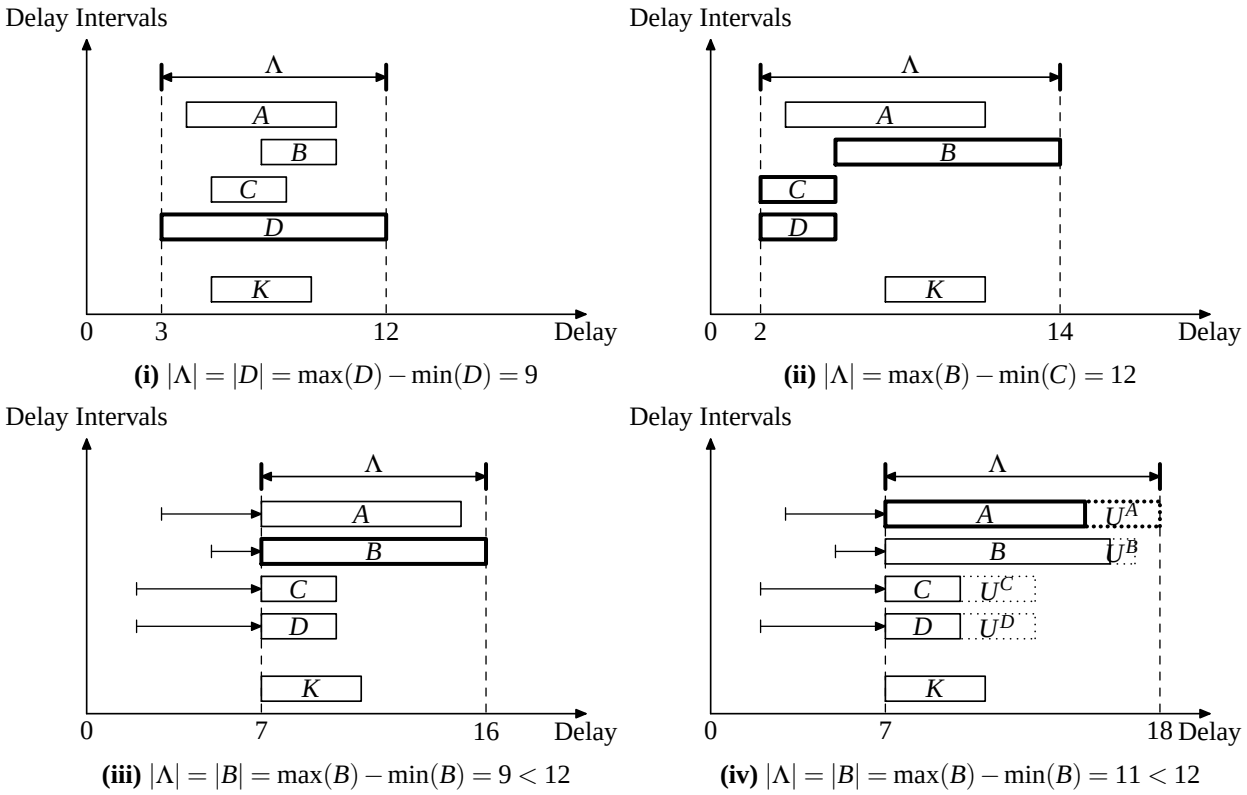


Figure 30: Two reconvergent data path systems satisfying (P1) and (P2), respectively.

the delay inserted on the path defining delay interval C in case (iii) of Figure 30 can be any value between 6 and 12 time units ($|C| = 3$) to satisfy the computed minimum interval. Similarly, the delay values inserted on all paths can simultaneously be increased by any identical amount (e.g. x time units) to generate an alternative solution. This non-unique solution set property provides a certain range of safety against any inherent uncertainty or unavailability of exact values of the delay elements.

The second important point to note is that after delay insertion, the interval lengths are preserved only if the inserted delay elements have no delay uncertainty. In demonstrating case (ii) of Figure 30, delay values with no uncertainties are considered in order to simplify the presentation of the delay insertion method. In reality, delay elements have delay uncertainties just like any other circuit component. These delay uncertainties of the delay

elements are accrued over the associated delay intervals. Let the *delay uncertainty* of the delay element inserted on path L be represented by U^L. The application of delay insertion to the sample system presented in case (ii) of Figure 30, where the delay uncertainties of the delay elements are accounted for, is presented in case (iv) of Figure 30. Note that due to the differences in the accrued delay uncertainties for each delay interval, the interval determining the minimum possible length for interval Λ can be different compared to the ideal case presented in case (iii). Incidentally, for cases (iii) and (iv) of Figure 30, the delay intervals determining the minimum possible length for Λ are B and A, respectively. Also, in a worst case scenario, the accrued delay intervals can end up being larger compared to the minimum length for Λ presented in case (ii). In the problem formulation presented later in Section 5.3, delay elements are realistically modeled with uncertainties.

Reflecting the proposition on a general reconvergent circuit, there are two possibilities in computing the minimum algebraic difference of (5.12):

P1*. The minimum and maximum system delays of the reconvergent data path system between R_d and R_c are determined by the same reconvergent path,

P2*. The minimum and maximum system delays of the reconvergent data path system between R_d and R_c are determined by two non-identical reconvergent paths.

For systems satisfying P1*, the minimum algebraic difference is already achieved. For systems satisfying P2*, delay insertion is used. By inserting delays in one or more of the reconvergent paths, the path with the largest difference between its maximum and minimum path delays *after delay insertion* becomes the determinant path for the minimum clock period T^*_{min} obtainable through delay insertion. Therefore, the minimum clock period of S_{FF} with clock skew scheduling and delay insertion is

$$T^*_{min} = \max_{\forall\alpha\in\{a,b\}} \left(PD^{12_\alpha}_M - PD^{12_\alpha}_m + U^{12_\alpha}\right) + S_2 + H_2. \tag{5.16}$$

Assuming zero delay uncertainty and substituting the numerical values, the minimum clock period T^*_{min} of S_{FF} after clock skew scheduling with delay insertion method is $T^*_{min} = 1.2 - 1.0 = 0.2$. The improvement achieved through delay insertion over circuits with clock skew scheduling is computed with the formula $[(T_{min} - T^*_{min})/T_{min}]100$. Substituting the values, the improvement is computed as $[(0.6 - 0.2)/0.6]100 = 66.7\%$.

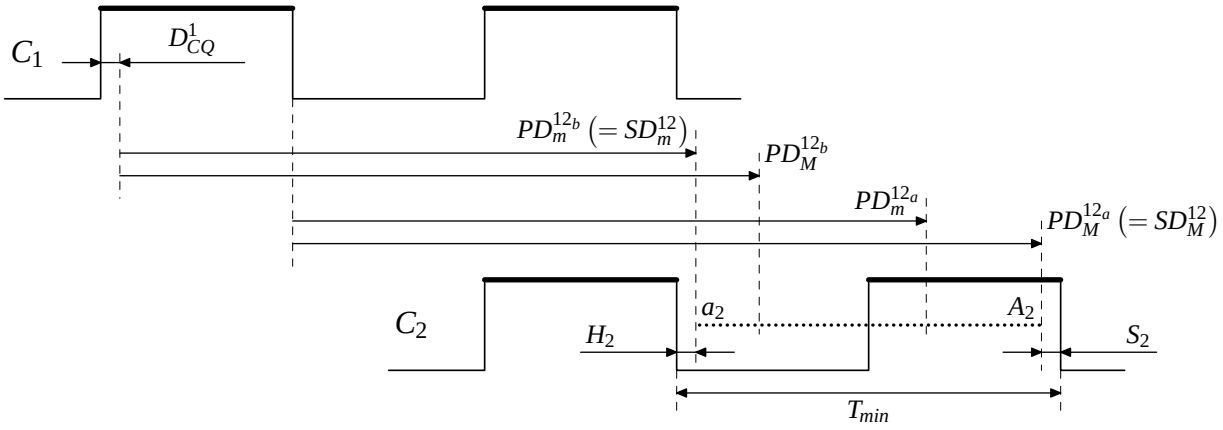


Figure 31: Timing of the simple level-sensitive reconvergent system in Figure 27 after CSS.

The computation of the amount of delays to be inserted on each path is integrated into the clock skew scheduling algorithm. For simplicity, continuous delay models are considered in here. The revised clock skew scheduling algorithm and initial insight for a general analysis using discrete delay models are presented in Sections 5.3 and 5.4.

5.2.3 Reconvergence in a Level-Sensitive Circuit

For level-sensitive circuits, results similar to an edge-triggered circuit are obtained despite the significant changes in circuit operation. The timing constraints are similar to the constraints for the edge-triggered circuit:

$$H_2 \leq a_2 \leq A_2 \leq T_{min} - S_2. \tag{5.17}$$

When clock skew scheduling is applied to S_L, the earliest and latest arrival times at R_2 satisfy

$$H_2 = a_2 \leq A_2 = T_{min} - S_2, \tag{5.18}$$

as illustrated in Figure 31. Using the same derivation as (5.7) and (5.10) and assuming $\left(D^1_{CQ} = D^1_{DQ}, d_1 = D_1\right)$ for practical reasons:

$$T_{min} = \max\left(PD^{12_a}_M, PD^{12_b}_M\right) - \min\left(PD^{12_a}_m, PD^{12_b}_m\right) + S_2 + H_2. \tag{5.19}$$

Substituting the numerical values and assuming zero internal register delays, the minimum clock period T_{min} of S_L after clock skew scheduling is $T_{min} = 0.6$.

The delay insertion method can also be used on level-sensitive circuits in order to improve the minimum clock period. The minimum clock period of S_L with clock skew scheduling and delay insertion is given by

$$T^*_{min} = \max_{\forall \alpha \in \{a,b\}} \left(PD^{12_\alpha}_M - PD^{12_\alpha}_m + U^{12_\alpha} \right) + S_2 + H_2. \tag{5.20}$$

The minimum clock period T^*_{min} of S_L after clock skew scheduling and delay insertion is computed as $T^*_{min} = 1.2 - 1.0 = 0.2$, leading to an improvement of 66.7% over circuit with clock skew scheduling. The revised clock skew scheduling algorithm for level-sensitive circuits is presented in Section 5.3.

Note that the earliest and latest data departure times d_1 and D_1, respectively, from a register R_1 can be non-identical in a level-sensitive circuit. Figure 31 illustrates one such case, where d_1 and D_1 occur at the leading and trailing edges of the clock signal, respectively. In such cases, the formulae in (5.19) and (5.20) do not hold true, however the minimum clock period remains directly proportional to the algebraic difference between the maximum and minimum path delays between R_1 and R_2. The delay insertion algorithm is fully applicable to all level-sensitive circuits, as the referred algebraic difference can ultimately be modified with delay insertion leading to improvements in the minimum clock period.

5.2.4 General Reconvergent Data Path Systems

The generalized case for a reconvergent data path system is presented in Figure 32. The edge-triggered and level-sensitive circuits are analyzed on the same circuit graph. Let there be k number of reconvergent paths between R_d and R_c, labeled $p_A, p_B, \ldots, p_K$. The generalized system contains $r^{d\{i_1 \ldots i_m\}c} = m$ and $r^{d\{j_1 \ldots j_n\}c} = n$ intermediate registers on two of its reconvergent paths, p_I and p_J, respectively ($p_I, p_J \in \{p_A, p_B, \ldots, p_K\}$). Assume that the minimum and maximum system delays between R_d and R_c are determined by paths $p^{d\{j_1 \ldots j_n\}c} = p_J$ and $p^{d\{i_1 \ldots i_m\}c} = p_I$, respectively. Note that, if $m \neq n$, the number of clock cycles for data propagation along the paths are different. After clock skew scheduling is

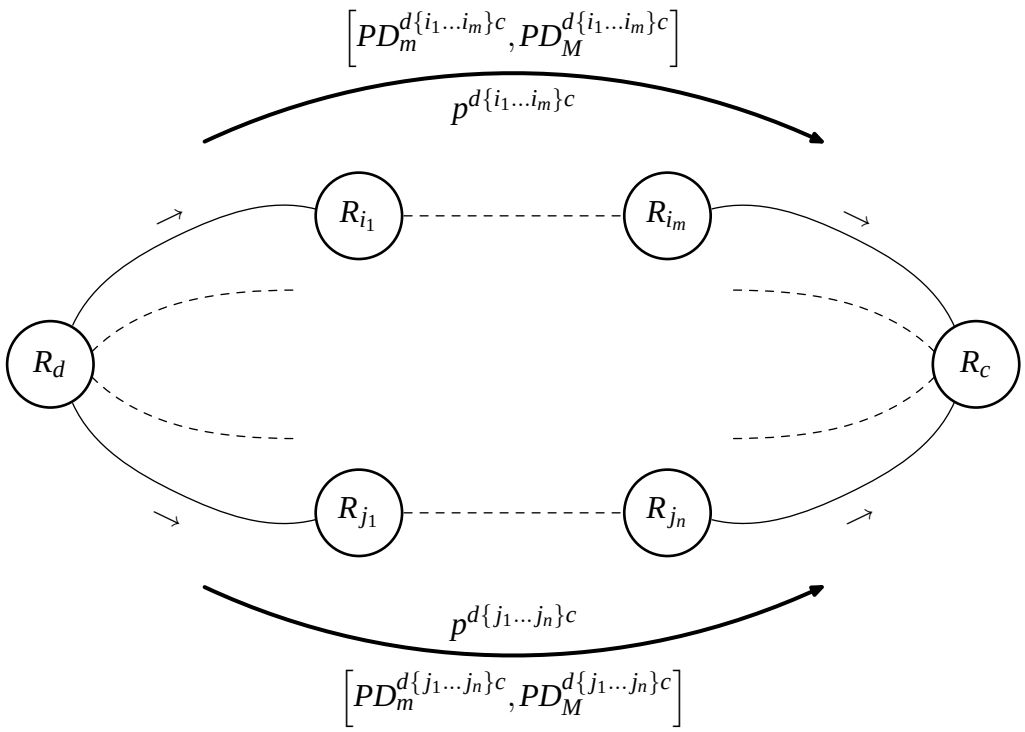


Figure 32: A generalized reconvergent data path system.

applied, the earliest and latest data arrival times at the convergent node with respect to the global zero time reference are

$$a_{c_{global}} = t_c + nT_{min} + H_c \tag{5.21}$$

$$A_{c_{global}} = t_c + (m+1)T_{min} - S_c. \tag{5.22}$$

Following from (5.7), the minimum clock period after clock skew scheduling is bounded by

$$|m-n+1|T_{min} = A_{c_{global}} + S_c - (a_{c_{global}} - H_c), \tag{5.23}$$

which leads to

$$T_{min} = \frac{PD_M^{d\{i_1 \ldots i_m\}c} - PD_m^{d\{j_1 \ldots j_n\}c} + S_c + H_c}{|m-n+1|} = \frac{SD_M^{dc} - SD_m^{dc} + S_c + H_c}{|m-n+1|}. \tag{5.24}$$

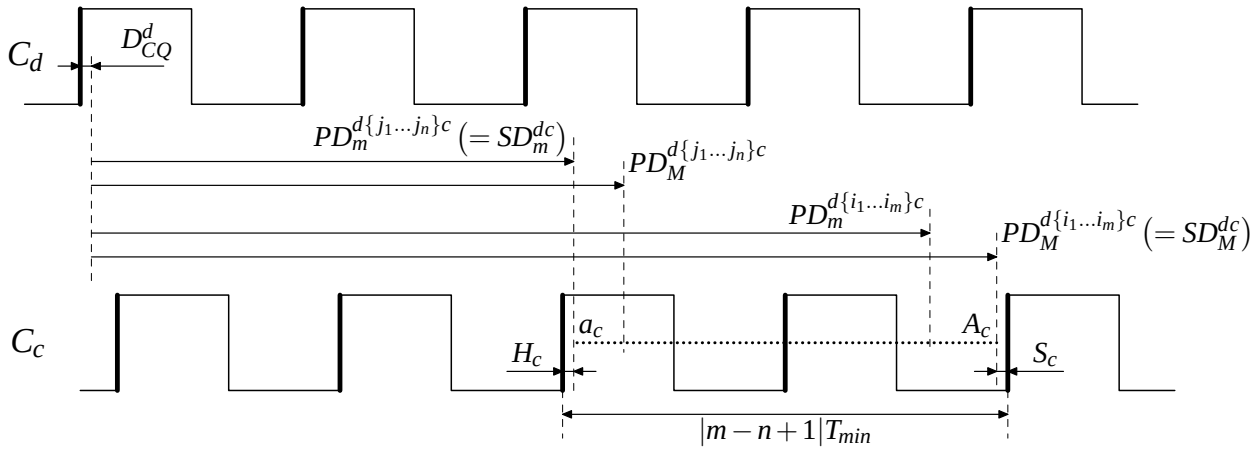


Figure 33: Timing of the edge-triggered reconvergent system with m=3 and n=2.

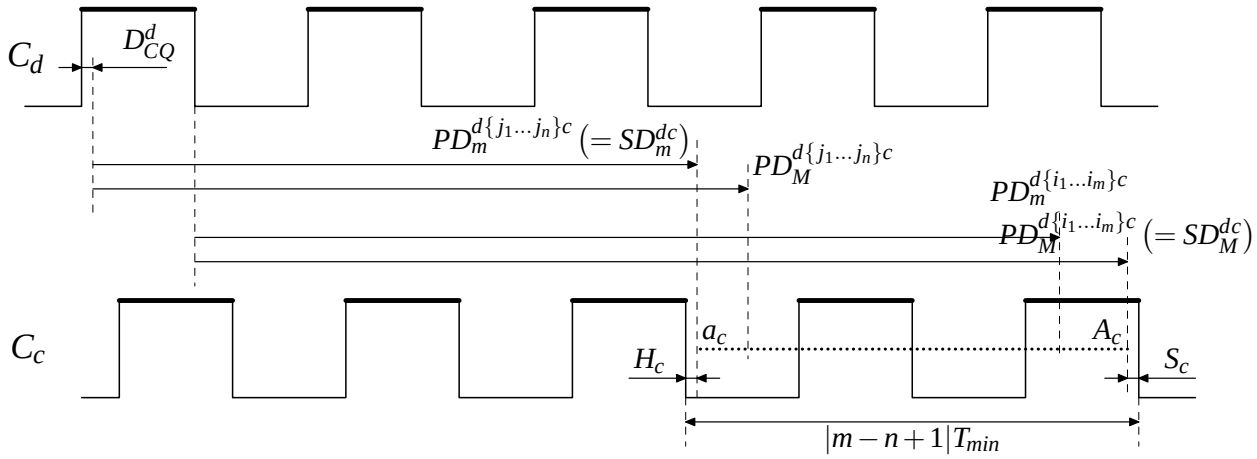


Figure 34: Timing of the level-sensitive reconvergent system with m=3 and n=2.

The identical lower bounds of the minimum clock period stated in (5.24) for both the edge-triggered and level-sensitive circuits are demonstrated in Figure 33 and Figure 34, respectively.

Similar to the simple reconvergence case analyzed in Section 5.2.1, if the minimum and maximum path delays are determined by the same reconvergent path, the delay insertion method is not beneficial. If these delays are determined by different reconvergent paths, the

delay insertion method is used to improve the minimum clock period. The minimum clock period achieved through clock skew scheduling and delay insertion is

$$T^*_{min} = \max_{\forall p_R, p_S \in \{p_A, p_B, \ldots, p_K\}} \left(\frac{PD_M^{p_R} - PD_m^{p_S} + U^{p_R} - U^{p_S}}{|m - n + 1|} \right) + \frac{S_c + H_c}{|m - n + 1|}. \tag{5.25}$$

Although the minimum path delay is not the total of the data propagation delays of the local data paths on the reconvergent path, the path delay can ultimately be modified by inserting delays on the local data paths. The amount of delay to be inserted is determined at run time by the clock skew scheduling algorithm.

5.3 FORMULATION AND ANALYSIS

In Sections 5.2.2 and 5.2.3, the limitation on the minimum clock period of a synchronous circuit with clock skew scheduling caused by a reconvergent system is shown for simple edge-sensitive and level-sensitive circuits. In Section 5.2.4, the limitation is parametrically calculated for a general representation of a reconvergent system. Note that the limitation caused by a reconvergent system defines the minimum clock period of a circuit only when this limitation is dominant over other limiting factors—the delay uncertainties of the data propagation times [21] and the total data propagation times on the data path cycles [56]. In this work, the details of the limiting factors other than the reconvergent paths are not analyzed in detail, as such limitations are well known.

A valid approach to computing the theoretical limitation caused by reconvergent paths in a synchronous circuit is to identify the reconvergent systems on a circuit graph and evaluate (5.25). Such an approach might not be trivial for certain circuit topologies (not discussed in this work). As a more practical approach, two generalized LP problems are defined in order to model the delay insertion method for level-sensitive and edge-sensitive synchronous circuits. These LP problems not only model and solve the clock period minimization problems, but also compute the optimal delay values to be inserted on each local data path in order to achieve the minimum possible clock period.

Table 7: CSS method for edge-sensitive circuits with the delay insertion method.

LP Model [21]	*LP Model [21] modified*
$\min T$	$\min T$
s.t. $T_{skew}(i,f) \leq T - D^{if}_{PM} - D^{i}_{CQM}$	s.t. $T_{skew}(i,f) \leq T - D^{if}_{PM} - D^{i}_{CQM} - I^{if}_{M}$
$T_{skew}(i,f) \geq -D^{if}_{Pm} - D^{i}_{CQm} + H_f$	$T_{skew}(i,f) \geq -D^{if}_{Pm} - D^{i}_{CQm} + H_f - I^{if}_{m}$
	$I^{if}_{M} \geq I^{if}_{m}$

Two clock skew scheduling algorithms presented in Table 2 and Table 6 for level-sensitive and edge-triggered circuits, respectively, are modified in order to integrate the delay insertion method. As reported in Chapter 4, LP models clock skew scheduling are highly amenable to accommodating additional design constraints.

The modified clock skew scheduling algorithms using the delay insertion method, assuming continuous delay models with uncertainty, are presented in Tables 7 and 8. The amount of delay to be inserted is formulated as the minimum-amount and maximum-amount variables I^{if}_{m} and I^{if}_{M}, respectively. Obviously, the uncertainty U^{if} of this delay element, defined in Section 5.2.2, is $U^{if} = I^{if}_{M} - I^{if}_{m}$. The delay variables are included in the propagation constraints on each local data path, however, pruning of the paths such that only the propagation constraints of the reconvergent paths are modified is also possible. For the former case, the clock skew scheduling algorithm simply returns zero for the delay values on the non-reconvergent paths.

5.4 PRACTICAL CONCERNS IN MODELING AND APPLICATION

In the problem formulation, continuous delay models have been used. Practically, however, delay elements are available only in discrete values. There are two possible approaches to solving the discrete valued delay insertion problem. The naive approach is to solve the clock

Table 8: CSS method for level-sensitive circuits with the delay insertion method.

LP Model	*LP Model modified*
min $T + M[\sum_{\forall j}(d_j + D_j) + \sum_{\forall j:\lvert FI(j)\rvert \geq 1}(A_j - a_j)]$	min $T + M[\sum_{\forall j}(d_j + D_j) + \sum_{\forall j:\lvert FI(j)\rvert \geq 1}(A_j - a_j)]$
s.t. $a_f \geq H_f$	s.t. $a_f \geq H_f$
$A_f \leq T - S_f$	$A_f \leq T - S_f$
$d_i \geq a_i + D^i_{DQm}$	$d_i \geq a_i + D^i_{DQm}$
$d_i \geq T - C^L_W + D^i_{CQm}$	$d_i \geq T - C^L_W + D^i_{CQm}$
$D_i \geq A_i + D^i_{DQM}$	$D_i \geq A_i + D^i_{DQM}$
$D_i \geq T - C^L_W + D^i_{CQM}$	$D_i \geq T - C^L_W + D^i_{CQM}$
$a_f \leq d_{i_n} + D^{i_n f}_{Pm} + T_{skew}(i_n, f) - T, \forall n$	$a_f \leq d_{i_n} + D^{i_n f}_{Pm} + I^{i_n f}_{m} + T_{skew}(i_n, f) - T, \forall n$
$A_f \geq D_{i_n} + D^{i_n f}_{PM} + T_{skew}(i_n, f) - T, \forall n$	$A_f \geq D_{i_n} + D^{i_n f}_{PM} + I^{i_n f}_{M} + T_{skew}(i_n, f) - T, \forall n$
$A_f \geq a_f$	$A_f \geq a_f$
$D_f \geq d_f$	$D_f \geq d_f$
	$I^{if}_M \geq I^{if}_m$

skew scheduling problem assuming continuous delays and approximating the optimal values with the given set of discrete components. Although likely to produce reasonable results for simple cases, such linear approximations to integer problems do not always guarantee optimality [92]. As a more robust and ubiquitously valid approach, the problem can be formulated as a mixed integer programming (MIP) problem. Evidently, the expected run times for MIP problems are typically longer than LP problems of similar size (see Section 4.6).

Modeling and solving the problem with continuous delay models serve best to demonstrate the two main purposes of this work; Identifying the limitation caused by the reconvergent paths and demonstrating how to mitigate these limitations through the delay insertion method. By adapting continuous delay models, the theoretical limitations of reconvergent paths and the level of improvement through mitigation of this limitation are analyzed *independent* of any cell library. For practical implementation, MIP-based solution approaches discussed above, or similar methods, must be used.

Another practical concern for the delay insertion method is the area-aware delay insertion method proposed in [68]. In order to reduce the total area increase due to inserted delays, a delay buffer tree structure is proposed. In the buffer tree structure, a shared delay element is placed between the fanouts—or fanins—of a register, if multiple fanouts of the same register

must be padded. Note that the delay buffer-tree construction is a post-timing analysis process and is not integrated into the clock skew scheduling algorithms.

Similar to Chapter 4, the local data paths are modeled abstractly at a higher hierarchy level than gate-level hierarchy. Such simplification is preferred in order to improve the demonstration of the theoretical limitation of reconvergent paths and the mitigation of this limitation by the delay insertion method. In practical implementation, the location of the delay elements to be inserted into the logic must be identified at a lower level of abstraction—most suitably at the gate-level of hierarchy. The modeling of local data paths at a higher abstraction level as suggested in this work might lead to an ambiguous assignment of delays to reconvergent paths. In an extreme case, it is plausible that three or more reconvergent paths might share all of the logic paths that constitute a reconvergent system. For the simplest case of four reconvergent paths, any two reconvergent paths might differ by one logic path only, and all logic paths might be covered by the four reconvergent paths. For such a reconvergent system, including delay elements anywhere on a reconvergent path (on any logic path) would affect the path delay of more than one reconvergent path. Thus, the optimal delay insertion values computed by the presented LP problem must be post-processed for practical implementation.

The described concerns in the practical implementation of the delay insertion method are not considered in the experimentation stage of this work. Simplicity is preserved in the models used in formulation in order to improve the presentation of the limitation caused by the reconvergent paths and the mitigation of this limitation by the delay insertion method. Designers, however, must be wary of these practical requirements.

5.5 EXPERIMENTAL RESULTS

For experimentation, the clock skew scheduling algorithms with the delay insertion method proposed for edge-triggered and level-sensitive circuits (Tables 7 and 8) are applied to the ISCAS'89 benchmark circuits. Continuous delay models have been used in the experimentation. The experimental setup in Chapter 4 is replicated for the proposed timing analyses.

The timing information for each circuit component is generated with the algorithm used in the experimentation section of Chapter 4, where the number of fanouts from a component, the size and the type of the component are considered effective on the computed delay. A 50% duty cycle single phase clock signal is selected. The internal register delays are assumed to be zero $\left(S_f = H_f = D^i_{CQ} = D^i_{DQ} = 0\right)$. The results computed on a 440MHz Sun Ultra-10 workstation with the barrier optimizer of the industrial LP solver CPLEX (version 7.5) [36] are presented in Tables 9 and 10. In Tables 9 and 10, the data shown are the number of registers r and paths p, the clock period T_{FF} for zero skew circuit with flip-flops, T^{CSS}_{FF} for non-zero skew circuit with flip-flops, and, T^{DICSS}_{FF} for non-zero skew circuit using delay insertion with flop-flops. Also listed are the calculation times t^{CSS}_{FF}, t^{DICSS}_{FF}, of T^{CSS}_{FF}, T^{DICSS}_{FF}, respectively, and the percentage clock period improvements I^{CSS}_{FF}, I^{DICSS}_{FF} and I^{DI}_{FF} for improvements from T_{FF} to T^{CSS}_{FF}, T_{FF} to T^{DICSS}_{FF}, T^{CSS}_{FF} to T^{DICSS}_{FF}, respectively.

The clock skew scheduling algorithms used in experimentation are targeting the clock period minimization problem. Therefore the improvements achieved in the minimum clock period through the application of clock skew scheduling and delay insertion methods are reported in Tables 9 and 10. These improvements are computed with the formula $\{I(\%) = [(T_{old} - T_{new})/T_{old}]100\}$. The zero clock skew, edge-sensitive synchronous circuit is selected as the common comparison mark due to its simplicity and popularity in digital circuit design. Both for edge-triggered and level-sensitive circuits, the improvements through conventional clock skew scheduling (I^{CSS}_{FF} and I^{CSS}_{L}, respectively) and through clock skew scheduling with delay insertion (I^{DICSS}_{FF} and I^{DICSS}_{L}, respectively) are computed. Also shown in Tables 9 and 10 are the comparisons of the non-zero clock skew circuits scheduled with conventional clock skew scheduling methods with non-zero clock skew circuits with delay insertion. These comparisons (I^{DI}_{FF} and I^{DI}_{L}, respectively, for edge-triggered and level-sensitive circuits) demonstrate the effectiveness of the delay insertion method in further improving the performance of a conventional clock skew scheduled circuit.

For the ISCAS'89 benchmark circuits, the delay insertion method leads to 10% and 9% improvements on average over the conventional clock skew scheduling algorithms for edge-triggered and level-sensitive circuits, respectively. For better visualization, the performance improvements in minimum clock period of edge-triggered and level-sensitive circuits achieved

Table 9: Delay insertion results for edge-sensitive ISCAS'89 benchmark circuits.

Edge-Triggered Circuits										
Circuit Info			Clock Periods (tu)			Run Times (s)		Improvements (%)		
Circuit	r	p	T_{FF}	T_{FF}^{CSS}	T_{FF}^{DICSS}	t_{FF}^{CSS}	t_{FF}^{DICSS}	I_{FF}^{CSS}	I_{FF}^{DICSS}	I_{FF}^{DI}
s27	3	4	6.6	4.1	4.1	0	0	38	38	0
s208.1	8	28	12.4	4.9	1.6	0	0	60	87	67
s298	14	54	13.0	9.4	9.4	0	0	28	28	0
s344	15	68	27.0	18.4	18.4	0	0	32	32	0
s349	15	68	27.0	18.4	18.4	0	0	32	32	0
s382	21	113	14.2	8.5	6.0	0	0	40	58	29
s386	6	15	17.8	17.3	17.3	0	0	3	3	0
s400	21	113	14.2	8.6	6.0	0	0	39	58	30
s420.1	16	120	16.4	6.8	1.6	0	0	59	90	76
s444	16	113	16.8	9.9	7.9	0	0	41	53	20
s510	6	15	16.8	14.8	14.8	0	0	12	12	0
s526n	21	117	13.0	9.4	9.4	0	0	28	28	0
s641	19	81	83.6	61.9	57.8	0	0	26	31	7
s713	19	81	89.2	63.8	59.4	0	0	28	33	7
s820	5	10	18.6	18.3	18.3	0	0	2	2	0
s832	5	10	19.0	18.8	18.8	0	0	1	1	0
s953	29	135	23.2	18.3	18.3	0	0	21	21	0
s1196	18	20	20.8	10.8	7.8	0	0	48	63	28
s1423	74	1471	92.2	77.4	75.8	0	0	16	18	2
s1488	6	15	32.2	29.0	29.0	0	0	10	10	0
s1494	6	15	32.8	29.6	29.6	0	0	10	10	0
s5378	179	1147	28.4	22.0	22.0	0	0	23	23	0
s9234	228	247	75.8	54.2	54.2	1	1	28	28	0
s13207	669	3068	85.6	57.1	53.8	1	2	33	37	6
s15850	597	14257	116.0	83.6	83.6	5	19	28	28	0
s15850.1	534	10830	81.2	57.4	57.4	5	10	29	29	0
s35932	1728	4187	34.2	20.4	15.7	1	6	40	54	23
s38417	1636	28082	69.0	42.2	42.2	15	37	39	39	0
s38584	1452	15545	94.2	65.2	62.8	5	15	31	33	4
Average								28	34	10

Table 10: Delay insertion results for level-sensitive ISCAS'89 benchmark circuits.

Level-Sensitive Circuits											
Circuit Info			Clock Periods (tu)			Run Times (s)		Improvements (%)			
Circuit	r	p	T_L	T_L^{CSS}	T_L^{DICSS}	t_L^{CSS}	t_L^{DICSS}	I_L	I_L^{CSS}	I_L^{DICSS}	I_L^{DI}
s27	3	4	5.4	4.1	4.1	0	0	18	38	38	0
s208.1	8	28	8.6	5.2	1.6	0	0	31	58	87	69
s298	14	54	10.6	9.4	9.4	0	0	18	28	28	0
s344	15	68	18.4	18.4	18.4	0	0	32	32	32	0
s349	15	68	18.4	18.4	18.4	0	0	32	32	32	0
s382	21	113	10.3	8.5	6.0	0	0	27	40	58	29
s386	6	15	17.3	17.3	17.3	0	0	3	3	3	0
s400	21	113	10.4	8.6	6.0	0	0	27	39	58	30
s420.1	16	120	12.6	7.2	1.6	0	0	23	56	90	78
s444	16	113	12.4	9.9	8.0	0	0	26	41	53	20
s510	6	15	14.8	14.3	14.3	0	0	12	15	15	0
s526n	21	117	10.6	9.4	9.4	0	0	18	28	28	0
s641	19	81	66.2	61.9	57.8	0	0	21	26	31	7
s713	19	81	71.2	63.8	59.4	0	0	20	28	33	7
s820	5	10	18.3	18.3	18.3	0	0	2	2	2	0
s832	5	10	18.8	18.8	18.8	0	0	1	1	1	0
s953	29	135	21.2	18.3	18.3	0	0	9	21	21	0
s1196	18	20	16.0	7.8	7.8	0	0	23	63	63	0
s1423	74	1471	86.4	75.8	75.8	1	2	6	18	18	0
s1488	6	15	29.0	29.0	29.0	0	0	10	10	10	0
s1494	6	15	29.6	29.6	29.6	0	0	10	10	10	0
s5378	179	1147	23.2	22.0	22.0	1	2	18	23	23	0
s9234	228	247	64.8	54.2	54.2	2	4	15	28	28	0
s13207	669	3068	67.4	57.1	53.8	4	7	21	33	37	6
s15850	597	14257	92.8	83.6	83.6	23	44	20	28	28	0
s15850.1	534	10830	71.4	57.4	57.4	23	34	12	29	29	0
s35932	1728	4187	34.1	20.4	15.7	7	16	0	40	54	23
s38417	1636	28082	54.8	42.2	42.2	41	101	21	39	39	0
s38584	1452	15545	76.4	65.2	62.8	31	51	19	31	33	4
Average								17	29	34	9

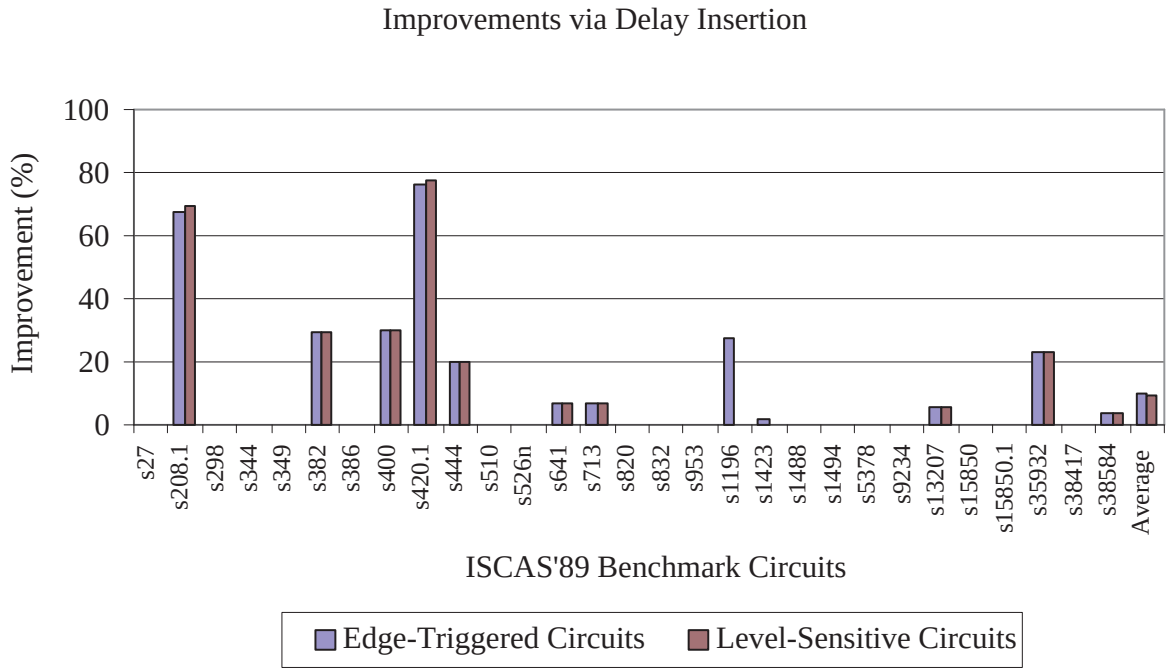


Figure 35: Percentage improvements through delay insertion in Tables 9 and 10.

respectively over corresponding non-zero clock skew edge-triggered and level-sensitive circuits are presented in Figure 35. Shown in Figure 35 are percentage improvements I_{FF}^{DI} and I_{L}^{DI} presented in Tables 9 and 10, respectively. Two data points shown per benchmark circuit from left-to-right are the improvements observed for edge-triggered and level-sensitive circuits, respectively. Note that these improvements are due to delay insertion simultaneous with clock skew scheduling.

The delay insertion method cannot be applied (not beneficial) to some circuits due to the two reasons discussed in Sections 5.2 and 5.2.2. The first reason, discussed in Section 5.2, is the fact that the minimum clock period of the circuit can be determined by a limitation other than reconvergent paths, which cannot be mitigated by the delay insertion method. The second reason, discussed in Section 5.2.2, is the fact that due to the uncertainty of the delay elements inserted into the logic, the delay insertion might be ineffective in improving the minimum clock period. In the LP formulations presented in Tables 7 and 8, the uncertainties

of the delay elements are modeled without lower (and upper) bounds (delay elements can have zero uncertainty with $I_m = I_M$). Thus, the second reason for inapplicability is not observed in the experimentation. Among the selected ISCAS'89 circuits, the delay insertion method for edge-triggered circuits is applicable to 41% (12 circuits) of the total 29 circuits. By excluding the circuits for which zero improvements are observed (for which the method is not applicable due to the first reason stated above), the average improvement of the delay insertion method for edge-triggered circuits is observed to be 26% over the conventional clock skew scheduling algorithm of [21]. The delay insertion method on level-sensitive circuits was applicable to 34% (10 circuits) of the total 29 circuits. By excluding the circuits for which zero improvements are observed, the average improvement of the delay insertion method for level-sensitive circuits is observed to be 27% on average over the conventional clock skew scheduling algorithm (Chapter 4).

The experimental results in Figure 35 show that reconvergent paths—with a significant probability (41% and 34% as observed on the ISCAS'89 circuits)—are the dominant limiting factor on the minimum clock period after clock skew scheduling for a synchronous circuit. The delay insertion method can effectively be used to mitigate these limitations, as shown by 26% and 27% improvements in the minimum clock period. The proposed clock skew scheduling method with delay insertion takes about twice as much time as the conventional application of clock skew scheduling, however, the method is highly practical with total run times below a few minutes with highly common computing resources.

The improvements in minimum clock period achieved through conventional clock skew scheduling (I_{FF}^{CSS} and I_{L}^{CSS}), and through clock skew scheduling with delay insertion (I_{FF}^{DICSS} and I_{L}^{DICSS}) for edge-triggered and level-sensitive circuits are visually presented for each benchmark circuit in Figures 36 and 37, respectively.

Shown in Figure 36 are the percentage improvements (I_{FF}^{CSS} and I_{FF}^{DICSS} in Table 9, respectively) in minimum clock period via clock skew scheduling and delay insertion for edge-triggered ISCAS'89 benchmark circuits. Two data points shown per benchmark circuit from left-to-right are the improvements observed for clock skew scheduling alone and delay insertion with clock skew scheduling, respectively. Shown in Figure 37 are the percentage improvements (I_{L}^{CSS} and I_{L}^{DICSS} in Table 10, respectively) in minimum clock period via clock

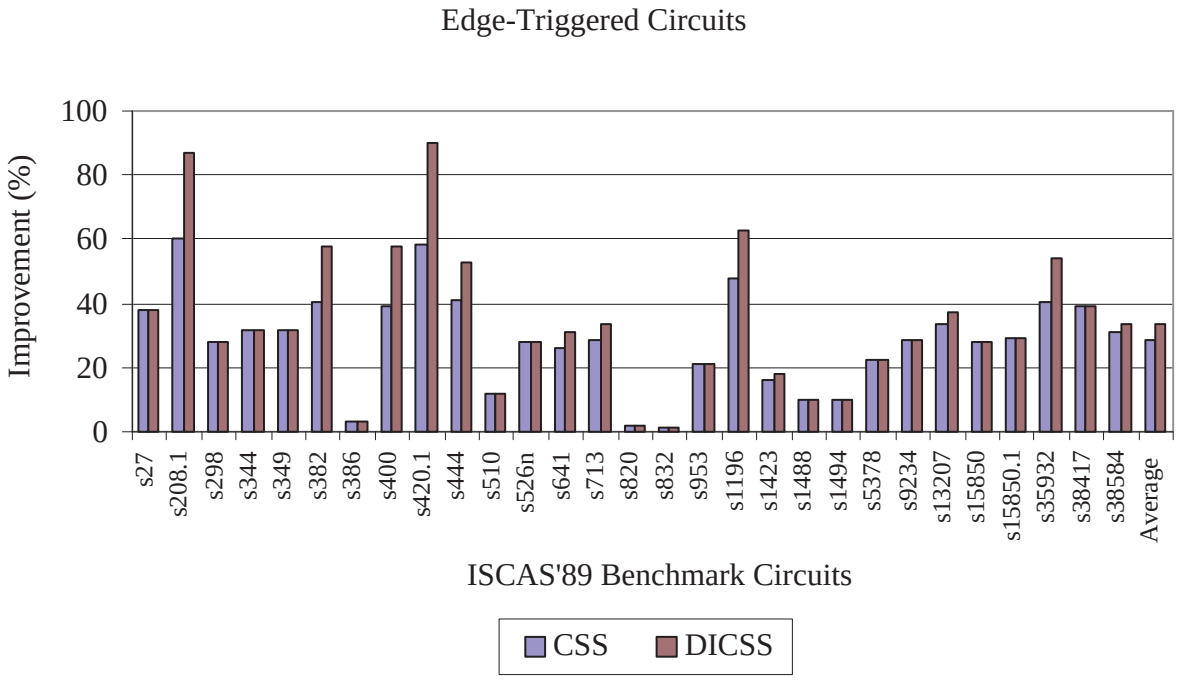


Figure 36: Percentage improvements on edge-triggered circuits in Table 9.

skew scheduling and delay insertion for level-sensitive ISCAS'89 benchmark circuits. Two data points shown per benchmark circuit from left-to-right are the improvements observed for clock skew scheduling alone and delay insertion with clock skew scheduling, respectively.

The average total improvement of non-zero clock skew, edge-triggered circuits with delay insertion with respect to the zero clock skew, edge-triggered circuits is 34%. The average total improvement of non-zero clock skew, level-sensitive circuits with delay insertion with respect to the zero clock skew, edge-triggered circuits is also 34%. Note that the total improvements are due to the simultaneous effects of the applications of delay insertion, clock skew scheduling and consideration of time borrowing (for level-sensitive circuits only) in the timing analysis. The improvement with delay insertion is equal to or greater than the improvement with clock skew scheduling only, as delay insertion is only applied when it can be used to mitigate the limitation of the reconvergent paths.

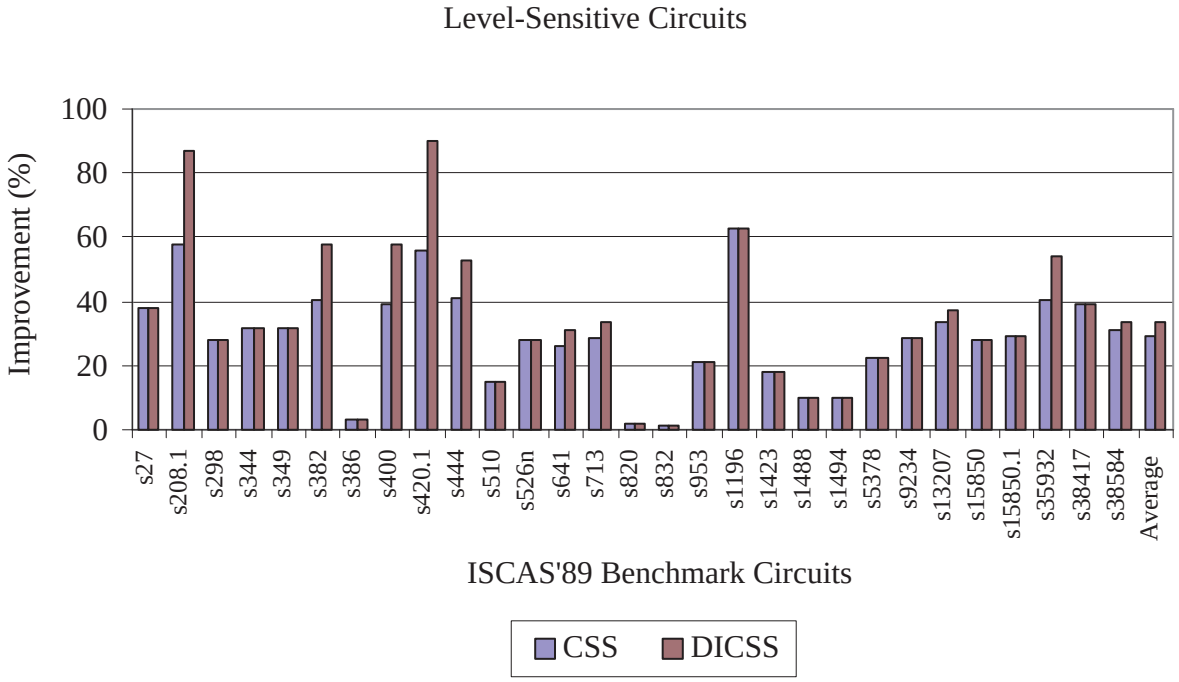


Figure 37: Percentage improvements on level-sensitive circuits in Table 10.

5.6 SUMMARY

In this chapter, the limitations of a reconvergent path on the improvements achievable through clock skew scheduling is shown for the first time. These limitations are mitigated with a novel delay insertion method. The delay insertion method is formulated as an LP problem, proposing a highly-automated, versatile and efficient implementation.

In experimentation, the delay insertion method is demonstrated to improve the minimum clock period after clock skew scheduling by up to 78% (9% and 10% on average for level-sensitive and edge-triggered circuits, respectively). The delay insertion method is demonstrated to be applicable to both edge-triggered and level-sensitive circuits with equally significant 34% (incidentally for the selected set of benchmarks, no theoretical conclusions are drawn) improvements on average in the minimum clock period (over traditional zero

clock skew, edge-triggered circuits). The practicality of delay insertion is to be concluded by circuit designers, considering the projected increases in the total power consumption and circuit area, and the transformation in circuit placement due to delay insertion.

The results of this research are important for the design and high-performance operation of non-zero clock skew circuits, especially due to the identification of the limitation caused by reconvergent data paths. Consequently, this research has been published in [80, 83, 85].

6.0 MULTI-PHASE NON-ZERO CLOCK SKEW SYNCHRONIZATION

Single-phase synchronization has traditionally been used in the design and analysis of systems, mostly due to its simplicity. Recently, however, multi-phase clock synchronization has become a necessity for larger and relatively complex integrated circuits. In order to provide multi-phase synchronization, conventional clock distribution networks have to be modified [33, 37, 44, 98]. Besides these conventional clock distribution networks modified for multi-phase synchronization, emerging clocking technologies such as resonant clocking technologies (discussed in Chapter 7) also encompass multi-phase synchronization schemes [96]. Such necessity to design and analyze for multi-phase synchronization schemes requires dedicated design and analysis frameworks.

In this chapter, an enhancement to the Linear Programming (LP) framework presented in Chapter 4 for non-zero clock skew, level-sensitive circuits is described in order to accommodate for multi-phase synchronization schemes. The enhanced framework is used to profile the performance improvement of level-sensitive circuits subject to clock skew scheduling under multi-phase synchronization. Time borrowing and clock skew scheduling are analyzed in single, two, three and four phase synchronization schemes. The effects of multi-phase synchronization schemes—independent of the clocking technology—on non-zero clock skew, level-sensitive circuit performance are analyzed.

In Section 6.1, a literature survey on the integration of multi-phase synchronization with non-zero clock skew circuit design is presented. The majority of the surveyed work is in the areas of circuit retiming and two-phase clocking. In Sections 6.2 and 6.3, the modeling and formulation of the timing analysis of a level-sensitive synchronous circuit are presented, respectively. In Section 6.4, the results of the clock period minimization problem for multi-phase synchronized circuits are analyzed. The chapter is summarized in Section 6.5.

6.1 PREVIOUS WORK

The advantages of level-sensitive design with multi-phase synchronization have previously been investigated in different contexts. One line of research has concentrated on circuit retiming, most notably in [56] and [18]. In [56], the advantages of two-phase, level sensitive circuits (as opposed to edge-sensitive circuits) are explored. It is concluded in [56] that the level of improvement in circuit performance is insignificant for such a circuit transformation, when circuit *retiming* is performed. In [18], the results of [56] are examined from a wider perspective, considering the depth of pipelining within a circuit—average improvements up to 30% are shown to be possible by two-phase, level-sensitive clocking with circuit retiming.

This proposed line of research differs from [56] and [18] by expanding the multi-phase synchronization concept to three, four and potentially higher number of phases (the studies presented in [56] and [18] are performed only for two-phase, level-sensitive circuits). Furthermore, the proposed research considers the application of clock skew scheduling as opposed to the application of circuit retiming in [56] and [18].

In [34], the authors advocate the use of a multi-phase clocking scheme for both edge-triggered and level-sensitive synchronous circuits for increased circuit performance. In [60], the number of clock phases constituting the multi-phase synchronization scheme and the skew values are restricted to reflect the practical limitations of conventional clock distribution networks. Neither [60] nor [34] aim to exhaustively discuss the effects of multi-phase synchronization on the level of improvement in circuit performance for non-zero clock skew, level-sensitive circuits.

6.2 MULTI-PHASE LEVEL-SENSITIVE CIRCUIT TIMING

The operation of a synchronous circuit previously defined in Chapter 4 for a single-phase synchronization scheme is redefined for a multi-phase synchronization scheme. In a generalized multi-phase synchronization scheme, the latches R_i and R_f of a local data path shown in Figure 38 are synchronized by the clock signals $C_i^{p_i}$ and $C_f^{p_f}$, respectively. The superscripts

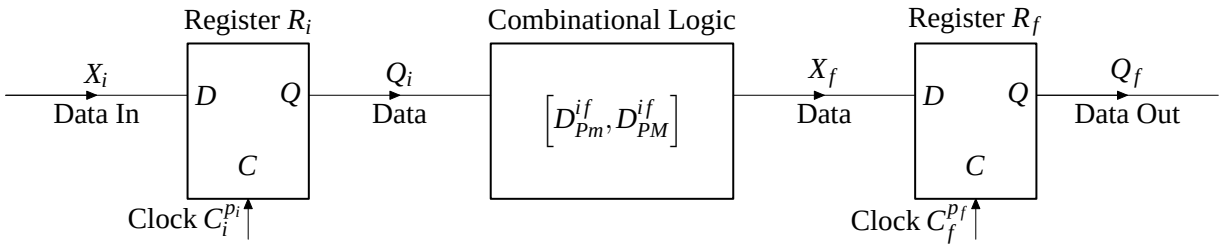
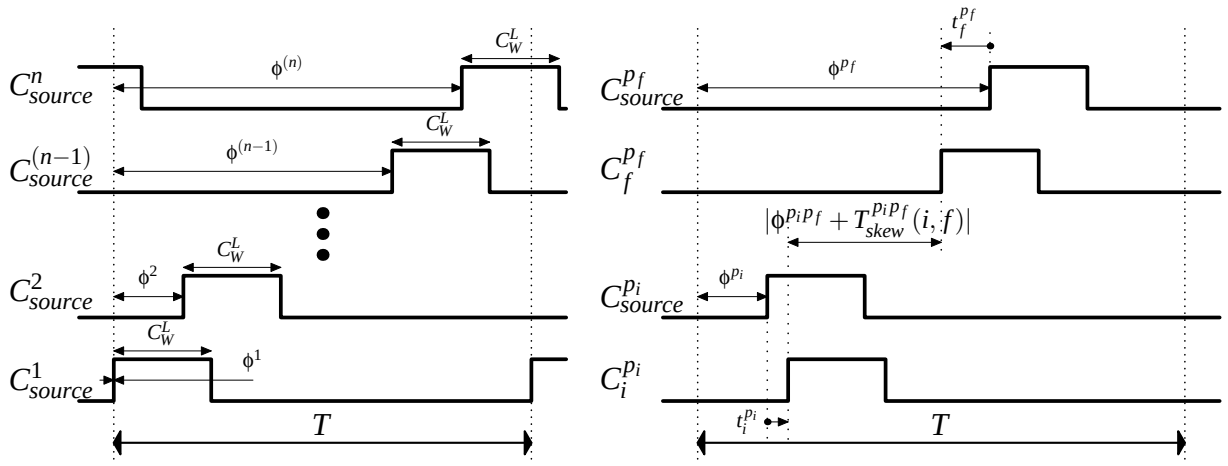


Figure 38: A local data path in a multi-phase synchronous circuit.

p_i and p_f describe the clock phases that synchronize R_i and R_f, respectively. The subscripts i and f denote the clock signals of phase C^{p_i} at R_i and phase C^{p_f} at R_f, respectively.

The generalized n-phase clocking scheme was introduced in Section 2.3. For convenience, the illustration of this generalized scheme is repeated in Figure 39(a). The set of clock signals $C^{global} = \{C^1, \ldots, C^n\}$ constitutes the n-phase clocking scheme. Note that $\{C^{p_i}, C^{p_f}\} \subset C^{global}$. The increase in the number of clock phases can lead to inaccuracies in the pulse width and clock edges for conventional clock distribution systems. In this work, there effects are disregarded under the assumption of the availability of improved or next-generation clocking technologies (such as the resonant clocking technologies presented in Chapter 7). Such an assumption is reasonable within CAD perspective adopted in this dissertation (see Section 2.4). An analysis bearing the presence of clock jitter is possible, yet does not marginally change the presented timing analysis framework. Practically, clock jitter can be modeled without any changes to the proposed framework, by assigning higher numerical values to the hold and setup-times of registers.

The *multi-phase clock skew* is defined as $T_{skew}^{p_i p_f}(i, f) = t_i^{p_i} - t_f^{p_f}$, where $t_i^{p_i}$ and $t_f^{p_f}$ are the delays of the clock signals $C_i^{p_i}$ and $C_f^{p_f}$ from the clock sources to the registers R_i and R_f, respectively. The multi-phase clock skew is illustrated in Figure 39(b). The common clock period for all clock phases is denoted by T for consistency with the original formulation of the single-phase synchronized circuits.



Clock Phases

(a)

Multi-Phase Clock Skew

(b)

Figure 39: Multi-phase clock and multi-phase clock skew.

In Figure 40, two local data paths starting at the latches R_{i_1} and R_{i_2}, respectively, and ending at R_f are considered. This figure is the multi-phase synchronization counterpart of Figure 14 shown on page 30. The clock signals driving the initial latches R_{i_1} and R_{i_2} are shown at the top and bottom, respectively. The middle clock signal corresponds to the final latch R_f. The time intervals for the arrival and departure times of latch data are illustrated by the upper and lower parallel dotted lines, respectively. Data delays are represented by the lengths of white or black rectangular boxes. Similar to the analysis in Chapter 4, the operational and constructional timing constraints of multi-phase, level-sensitive circuits are formulated based on these data propagation rules.

The timing constraints governing the operation of a multi-phase, level-sensitive synchronous system are summarized in Table 11. These constraints are valid for all varieties of overlapping and non-overlapping clocking schemes, and for any feasible selection of duty cycles per clock phase. Note the *max* and *min* functions in the synchronization and propagation constraints in Table 11. The non-linearities of these constraints are similar to those reported in Section 4.4 for single-phase circuits. Consequently, the multi-phase problem is solved by linearizing the non-linear constraints with the MBM method (Section 4.4.1).

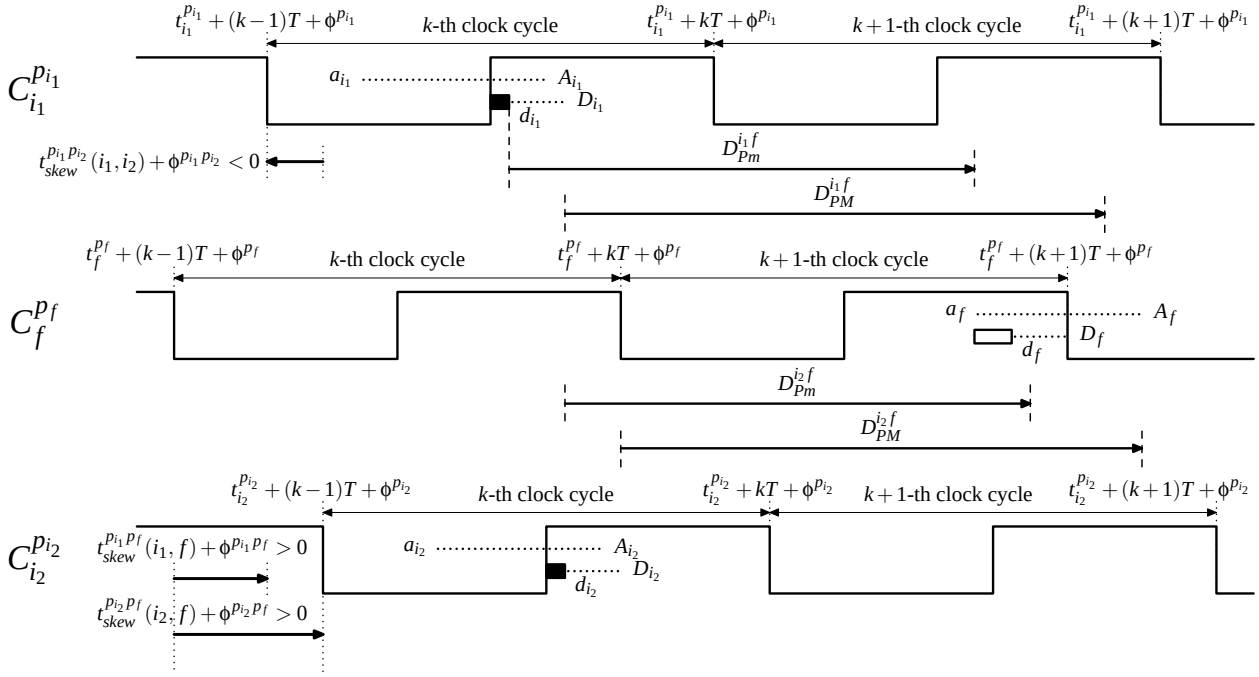


Figure 40: Propagation of the data signal in a simple multi-phase circuit.

Table 11: Operational timing constraints of a multi-phase level-sensitive circuit.

Latching	$a_f \geq H_f$ $A_f \leq T - S_f$
Synchronization	$d_i = \max\left(a_i + D^i_{DQm}, T - C^L_W + D^i_{CQm}\right)$ $D_i = \max\left(A_i + D^i_{DQM}, T - C^L_W + D^i_{CQM}\right)$
Propagation	$a_f = \min\limits_{\forall i_n}\left(d_{i_n} + D^{i_n f}_{Pm} + T^{p_{i_n}p_f}_{skew}(i_n,f) + \phi^{p_{i_n}p_f}\right)$ $A_f = \max\limits_{\forall i_n}\left(D_{i_n} + D^{i_n f}_{PM} + T^{p_{i_n}p_f}_{skew}(i_n,f) + \phi^{p_{i_n}p_f}\right)$

6.3 LINEARIZATION OF THE TIMING ANALYSIS

Similar to Section 4.4, an optimization problem is constructed, where the objective is to minimize the clock period T and the constraints are compiled from the operational constraints in Table 11 and the constructional constraints (Section 4.3). The resulting optimization problem is an NLP problem due to non-linearity of the synchronization and propagation constraints. The modified big M method (MBM method) (Section 4.4.1) is used to generate an efficient LP model formulation of this optimization problem. The LP model for the clock period minimization problem of a level-sensitive circuit synchronized by a multi-phase synchronization scheme is presented in Table 12. Similar to the discussion in Section 4.6, the optimality of the results for the LP formulation is verified using the exact MIP formulation with post-solution checks.

6.4 EXPERIMENTAL RESULTS

Experiments with the ISCAS'89 benchmark circuits are performed in order to observe and quantify the level of improvement achieved through clock skew scheduling and time borrowing on level-sensitive circuits with multi-phase synchronization. Multi-phase synchronization of ISCAS'89 benchmark circuits are performed using the transformation shown in Figure 41. This transformation is similar to the procedure used in the literature, particularly in [7, 8, 67, 73, 74]. The timing information of the benchmark circuits is generated with a similar algorithm to the one used in Chapters 4 and 5, where the type, size and fanout of a gate are considered in the computed delays.

The experiments are performed using two, three and four-phase clocking schemes representing various degrees of multi-phase synchronization. For simplicity, non-overlapping multi-phase clock signals with identical duty cycles, shown in Figure 42, are used in experimentation. Due to the transformation shown in Figure 41, a new level of latches is required for each additional clock phase. The latches are modeled with inherent delays in order to capture these effects in formulation. Latches are modeled with a delay pair of [0.9, 1.1] time

Table 12: LP model clock skew scheduling problem of multi-phase level-sensitive circuits.

LP Model	
min	$T + M[\sum_{\forall R_j}(d_j + D_j) + \sum_{\forall R_k:\|Fan-In(R_k)\|\geq 1}(A_k - a_k)]$
subject to	
Latching-Hold time	$a_f \geq H_f$
Latching-Setup time	$A_f \leq T - S_f$
Synchronization-Earliest time	$d_i \geq a_i + D^i_{DQm}$ $d_i \geq T - C^L_W + D^i_{CQm}$
Synchronization-Latest time	$D_i \geq A_i + D^i_{DQM}$ $D_i \geq T - C^L_W + D^i_{CQM}$
Propagation-Earliest time	$a_f \leq d_{i_n} + D^{i_n f}_{Pm} + T^{p_{i_n} p_f}_{skew}(i_n, f) + \phi^{p_{i_n} p_f}, \forall R_{i_n}$
Propagation-Latest time	$A_f \geq D_{i_n} + D^{i_n f}_{PM} + T^{p_{i_n} p_f}_{skew}(i_n, f) + \phi^{p_{i_n} p_f}, \forall R_{i_n}$
Validity-Arrival time	$A_f \geq a_f$
Validity-Departure time	$D_f \geq d_f$
Initialization	$A_l = d_l - (D^l_{CQm}$ or $D^l_{DQm}), \forall R_l : \|Fan - In(R_l)\| = 0$

units, corresponding to the minimum and maximum delays for a latch. For reference, a unity delay (a delay of 1 time unit) is close to the delay value of an FO4 inverter in the proposed delay generation algorithm.

For multi-phase synchronization, each additional clock phase requires a new level of latches to be inserted into data paths, effectively increasing path delays. Thus, as the number of clock phases increases, the performance of a zero clock skew system degrades. For non-zero clock skew systems, however, this is not necessarily the case as shown by these experiments. The solutions of clock period minimization problems computed with CPLEX (v7.5) barrier optimizer [36] on a 440MHz Sun Ultra-10 workstation are presented in Tables 13, 14 and 15, and Figures 43, 44 and 45. The number of registers r and paths p (before modification) of the

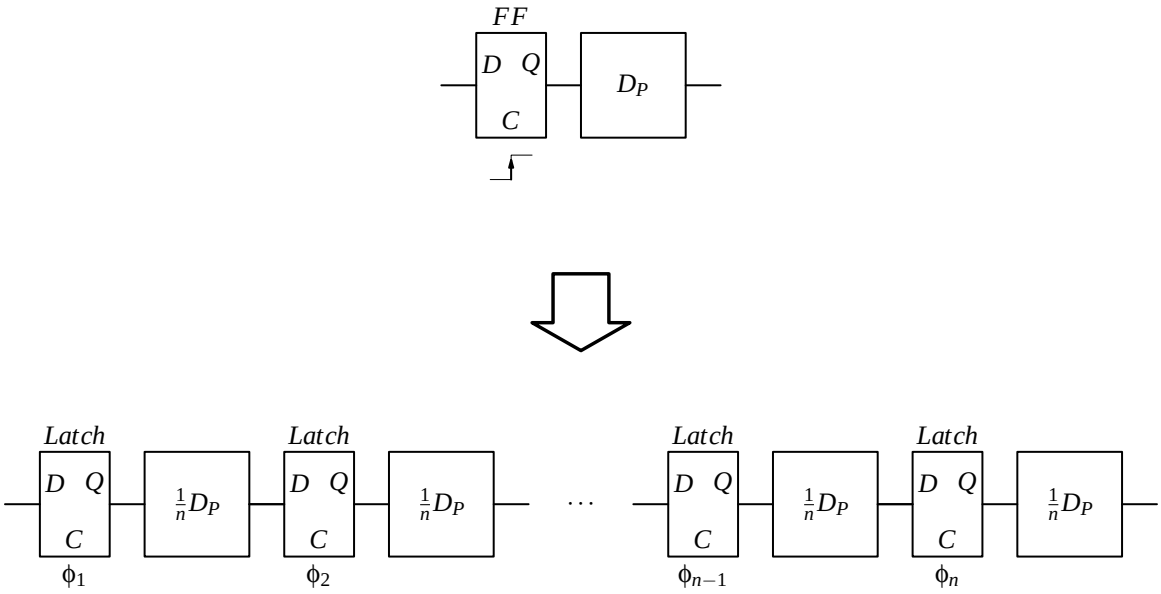


Figure 41: Generation of an n-phase data path with latches.

ISCAS'89 benchmark circuits are shown in Tables 13, 14 and 15 (on pages 110, 111 and 112, respectively). Minimum clock periods, improvements and calculation time are denoted by T, I and t, respectively. Subscripts FF, $n\phi$ represent circuit topologies for flip-flop based and n-phase level-sensitive circuits, respectively. Superscripts and titles $TB, CSS, TBCSS$ stand for time borrowing, clock skew scheduling and both, respectively. Minimum clock periods (T) are measured in time units.

In the rest of this section, the experimental results and factors contributing to the improvements in these results are discussed in greater detail. In particular, the properties of multi-phase synchronization which affect level-sensitive circuit performance are discussed in Section 6.4.1. The effects of multi-phase synchronization on time borrowing are addressed in Section 6.4.2. The effects of multi-phase synchronization on clock skew scheduling are addressed in Section 6.4.3. Finally, the effects of multi-phase synchronization on the simultaneous application of time borrowing and clock skew scheduling are addressed in Section 6.4.4.

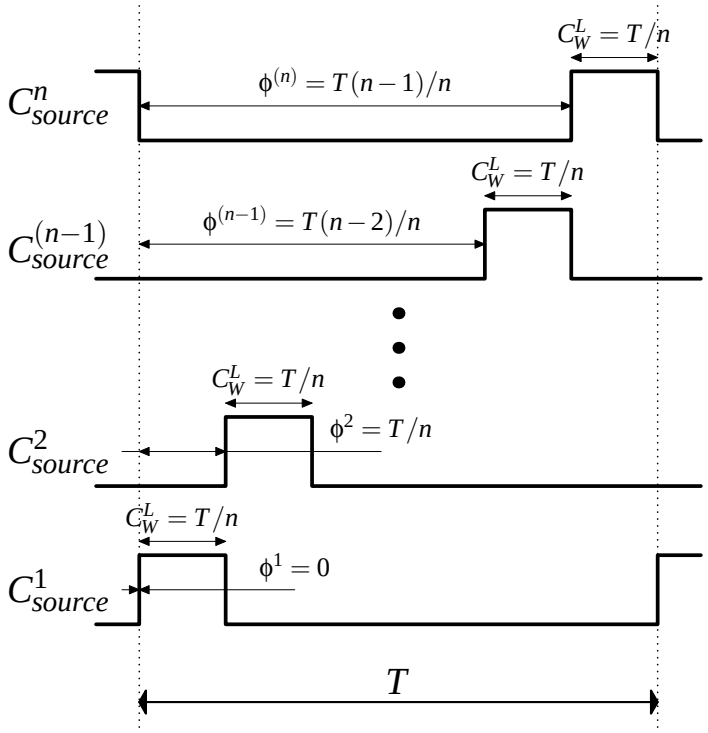


Figure 42: Non-overlapping multi-phase synchronization clock.

6.4.1 Multi-Phase Clocking

Multi-phase clocking is superior to single phase clocking by better accommodating the transparency periods of latches. Depending on the particular synchronization scheme, however, the duration of the transparency periods can be short for each phase, thereby reducing the advantages of multi-phase clocking. In single-phase clocking, the transparency periods of latches have identical positions within their respective clock cycles. In multi-phase clocking, the transparency periods of different clock phases are distributed over the clock cycle. In a multi-phase circuit synchronized with the clock signal shown in Figure 42, for instance, the transparency periods are located at different times within the clock cycle (*e.g.*, clock phases C^1 and C^n are the first and last sections, respectively). Such variety in the locations of transparency periods provides flexibility on the permissible data propagation times of a

local data path. The assorted assignment of clock phases to registers, achieved through clock skew scheduling or any other methods, leads to improvements in the circuit performance.

As illustrated in the transformation procedure shown in Figure 41, an extra level of latches is inserted within a logic data path for each clock phase. The delays of these inserted latches can become significant for higher number of clock phases and degrade the circuit performance of multi-phase circuits in the absence of clock skew scheduling. For non-zero clock skew circuits, the negative effects of latch insertion are compensated for and offset, often leading to an overall improvement in circuit performance.

The transformation procedure in Figure 41 leads to a certain bias in circuit operation, such that, each sequentially adjacent latch pair is synchronized by consecutive clock phases. Furthermore, the propagation delays of the combinational blocks are distributed evenly between clock phases according to the transformation procedure. In practical application of the presented procedure, the particular clock phase and propagation delay information of a circuit must be reflected in the formulation.

6.4.2 Multi-Phase Clocking Effects on Time Borrowing

In order to observe the effects of multi-phase clocking on time borrowing, each flip-flop in the benchmark circuit is replaced with a latch (n latches in the case of n-phase clocking). As shown in Table 14, an average improvement of 16.7% is achieved for single-phase circuits. Also, average improvements of 15.3%, 8.0% and 1.6% are achieved for two, three and four phase clocking schemes, respectively. For a visual representation of these results, the percentage improvements presented in Table 14 for each benchmark circuit are illustrated in Figure 43. In Figure 43, four data points shown per benchmark circuit from left-to-right are the percentage improvements observed for the single-phase, two-phase, three-phase and four-phase synchronization schemes, respectively.

It is observed that the improvement achieved through time borrowing decreases as the number of clock phases increases. This degradation is expected, because by definition, the transparency period of latches shortens for higher number of clock phases. The degradation is also caused by the increased significance of the inserted latch delays for higher number of

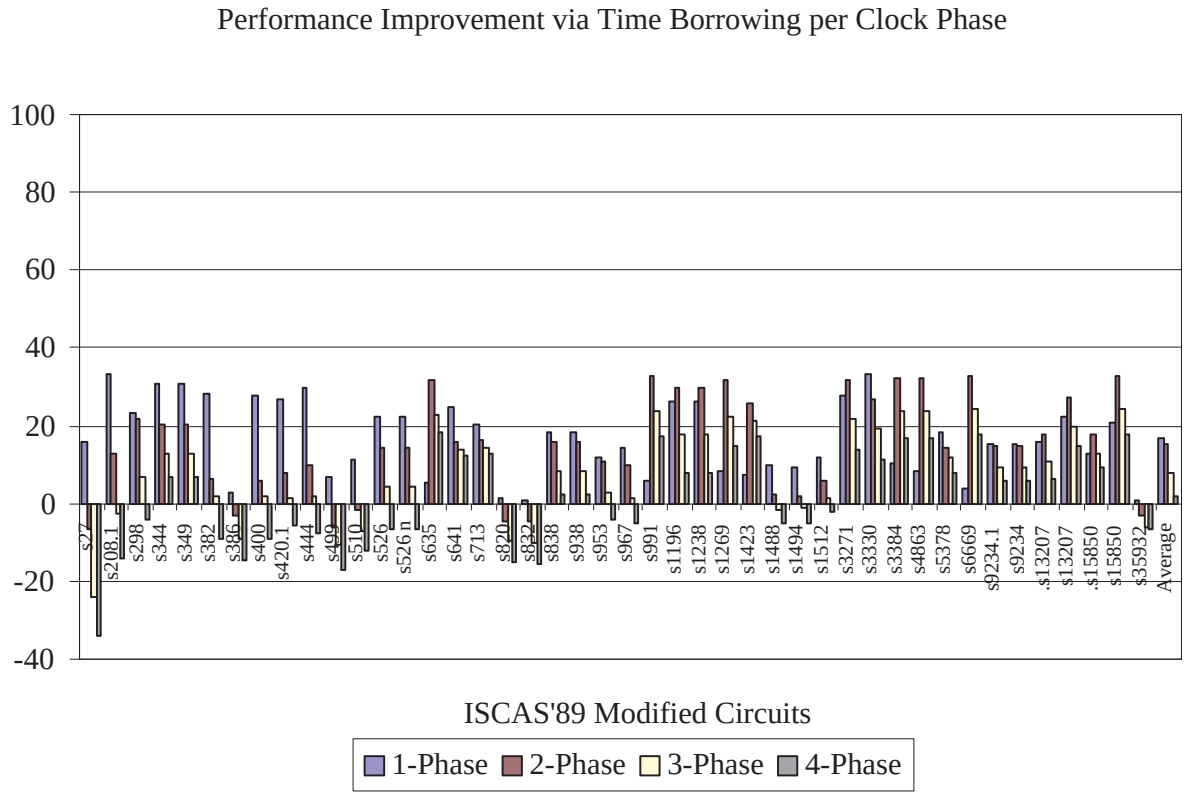


Figure 43: Effects of multi-phase clocking on time borrowing.

clock phases in contributing to the total path delay. With shortened transparency periods on latches, elongated path delays (quantitatively or relative to the transparency period) reduce the slack time on data paths, suppressing the improvements achieved through time borrowing.

It is also important to note that synchronization with a multi-phase clocking scheme has similar effects with time borrowing on circuit operation. In other words, the slack propagation time is shared between the multi-phase clocks, which would otherwise be utilized by time borrowing. This fact also contributes to the degradation of improvement for multi-phase synchronization.

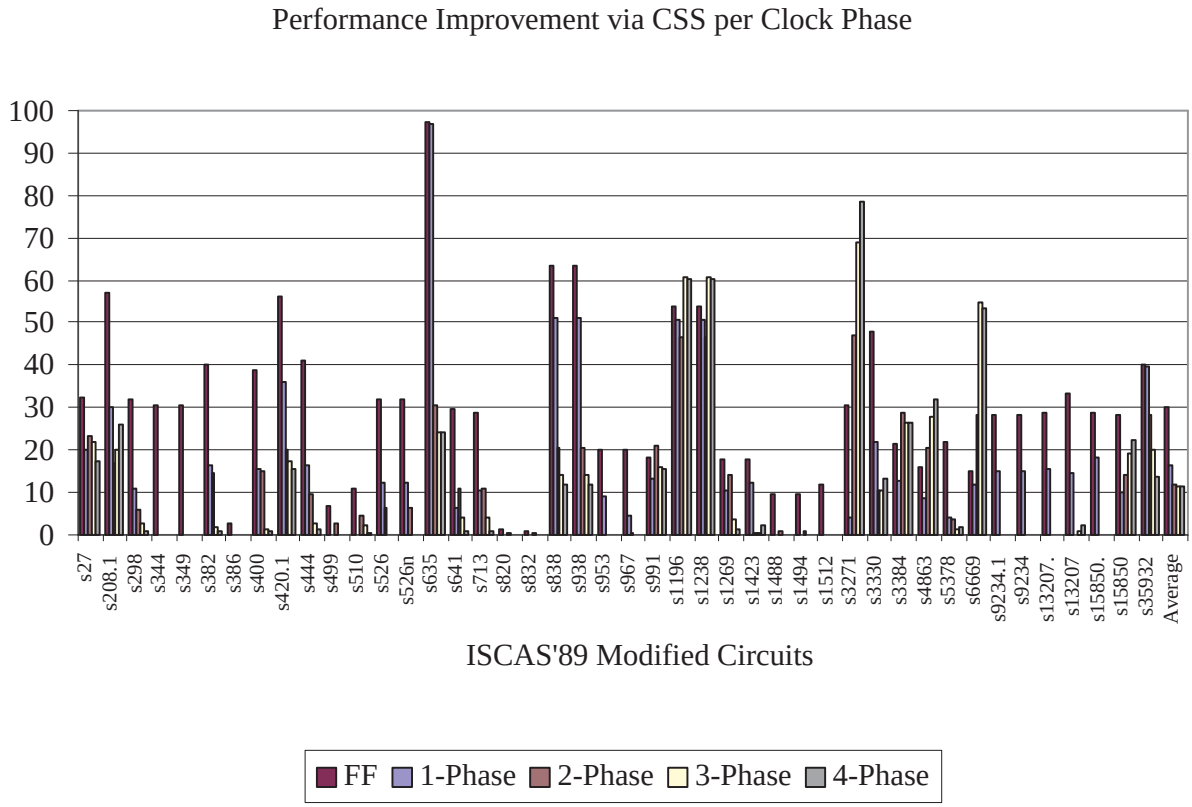


Figure 44: Effects of multi-phase clocking on clock skew scheduling.

6.4.3 Multi-Phase Clocking and Clock Skew Scheduling

The results for edge-sensitive circuits display improvements of 30.3% on average due to clock skew scheduling *alone*. For single, two, three and four-phase level-sensitive implementations, clock skew scheduling results in 16.5%, 12.1%, 11.4% and 11.3% improvements on average, respectively. The percentage improvements for each benchmark circuit are illustrated in Figure 44. In Figure 44, five data points shown per benchmark circuit from left-to-right are the percentage improvements observed for the edge-sensitive, single-phase, two-phase, three-phase and four-phase synchronization schemes, respectively. The degradation in performance (compared to non-zero clock skew, edge-sensitive circuits) is expected as the even distribution

of the transparency periods—due to the multi-phase clocking scheme of choice in this work—potentially negates the effectiveness of clock skew scheduling.

Clock skew scheduling is more effective on circuits with certain characteristics. In particular, if the data propagation delays on the local data paths of a circuit are irregular, higher improvements in the circuit performance are achievable through clock skew scheduling. The transformation procedure shown in Figure 41 proposes an even distribution of data propagation delays for adjacent local data paths, increasing the regularity of the circuit for increasing number of clock phases. Therefore, the high regularity of the multi-phase circuits—due to the bias in the transformation procedure in Figure 41—also contributes to the degradation.

6.4.4 Simultaneous Time Borrowing and Clock Skew Scheduling

For non-zero clock skew circuits, *total* improvements of 30.3%, 24.8%, 17.7% and 12.0% on average are observed for single, two, three and four phase clocking schemes, respectively. These total improvements are due to the *simultaneous* application of clock skew scheduling and time borrowing. The percentage improvements for each benchmark circuit are illustrated in Figure 45. In Figure 45, four data points shown per benchmark circuit from left-to-right are the percentage improvements observed for the single-phase, two-phase, three-phase and four-phase synchronization schemes, respectively.

As discussed in Sections 6.4.2 and 6.4.3, the improvements achieved through time borrowing and clock skew scheduling individually decrease as the number of clock phases increases. Nevertheless, the observed improvements are generally superior compared to zero clock skew, edge-sensitive circuits (exceptions are the benchmarks for which negative improvements are reported—mostly due to inserted latch delays). Exemplifying the positive trend is the analysis of the benchmark circuit $s1196$, for instance, where an improvement of 68% is observed for three-phase clocking through time borrowing and clock skew scheduling. For the same circuit, the improvements are at 63%, 63% and 64% for single, two and four-phase clocking, respectively.

It is observed in Tables 13 and 14 that no particular multi-phase approach is superior to others *in all cases*. Investigation of various approaches is found mandatory in order to

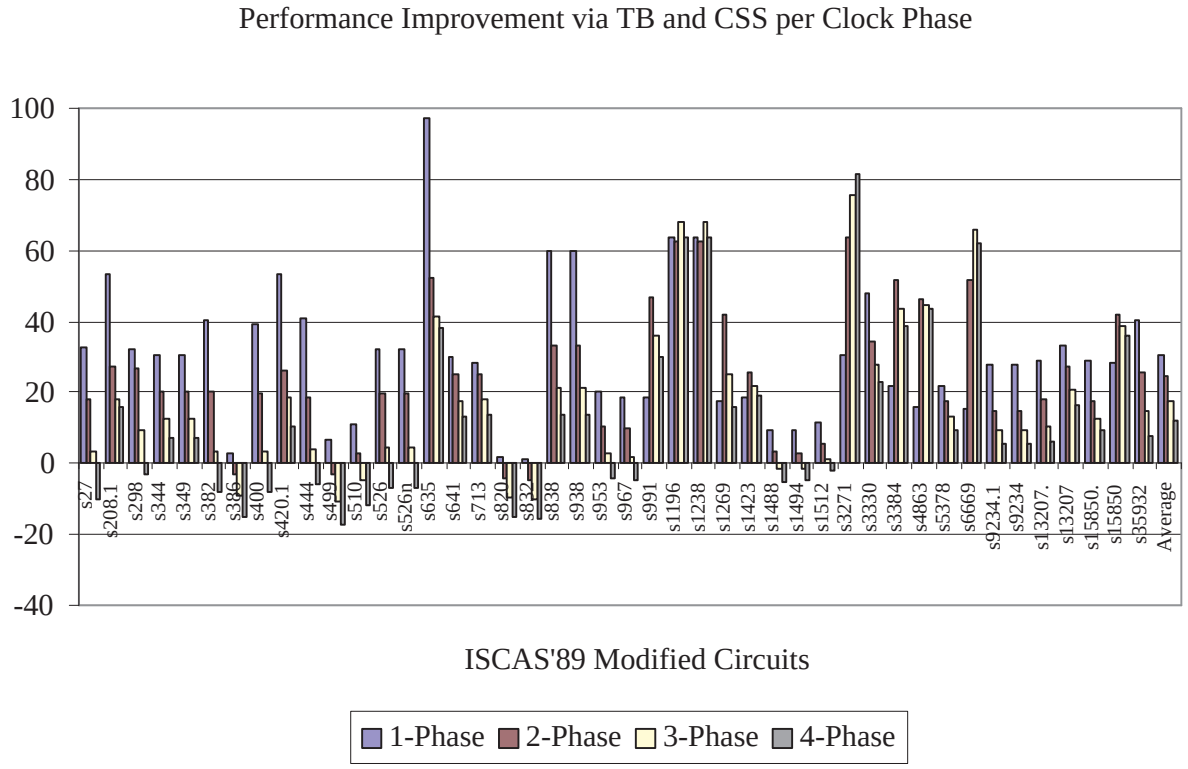


Figure 45: Effects of multi-phase clocking on time borrowing and clock skew scheduling.

identify the optimal synchronization scheme for any particular circuit (optimal in achieving an absolute minimum clock period).

6.5 SUMMARY

In this chapter, individual and simultaneous applications of clock skew scheduling and time borrowing to multi-phase, level-sensitive circuits are examined. By solving the efficient LP model formulations of the inherently non-linear clock period minimization problem, it is observed that when compared to single-phase circuits, multi-phase circuits generally benefit less from clock skew scheduling and time borrowing. Clock skew scheduling is more effective

on the improvement of edge-sensitive circuits due to the short on-time (transition edge) of synchronizing clock signals.

It is concluded that in the design of a level-sensitive synchronous circuit, if clock skew scheduling is applied and time borrowing is fully accounted for, multi-phase synchronization might not be necessary to achieve maximum circuit speed. However, exploration of all possible cases is mandatory in order to reach the optimal result because no particular synchronization scheme (number of phases) is consistently superior to others.

For clocking technologies, including the technologies where multi-phase synchronization and clock skew scheduling are essential (such as rotary clocking technology), a complete assessment of circuit performance is usually performed by considering all relevant criteria, not only clock speed. Among the most common criteria are total power dissipation, robustness to process parameter variations, scalability and underlying design complexity. The trade-off between the circuit performance and the implementation and analysis complexities arising from different synchronization alternatives must be evaluated for each particular design.

The CAD framework presented in this chapter is the first to correctly model and analyze the effects of multi-phase synchronization on non-zero clock skew circuit operation. In the experiments, it is shown for the first time that multi-phase synchronization of static circuits can actually be advantageous in terms of circuit speed, despite the increased path delays due to latch insertion per each additional clock phase. Such a fact is contrary to common wisdom, which has over the years been suggested for zero clock skew systems. The results of this research have been published in [79, 82, 84].

Table 13: Minimum clock periods of multi-phase ISCAS'89 benchmark circuits.

Circuit	**Zero Clock Skew**					**Non-Zero Clock Skew**				
Circuit	T_{FF}	$T_{1\phi}^{TB}$	$T_{2\phi}^{TB}$	$T_{3\phi}^{TB}$	$T_{4\phi}^{TB}$	T_{FF}^{CS}	$T_{1\phi}^{TBCSS}$	$T_{2\phi}^{TBCSS}$	$T_{3\phi}^{TBCSS}$	$T_{4\phi}^{TBCSS}$
s27	7.7	6.5	8.2	9.5	10.3	5.2	5.2	6.3	7.4	8.5
s208.1	13.5	9.0	11.8	13.9	15.4	5.8	6.3	9.8	11.1	11.4
s298	14.1	10.8	11.0	13.1	14.7	9.6	9.6	10.3	12.8	14.6
s344	28.1	19.5	22.4	24.5	26.1	19.5	19.5	22.4	24.5	26.1
s349	28.1	19.5	22.4	24.5	26.1	19.5	19.5	22.4	24.5	26.1
s382	15.3	11.0	14.4	15.0	16.7	9.2	9.2	12.2	14.8	16.5
s386	18.9	18.4	19.5	20.6	21.7	18.4	18.4	19.5	20.6	21.7
s400	15.3	11.1	14.4	15.0	16.7	9.3	9.3	12.3	14.8	16.5
s420.1	17.5	12.8	16.1	17.2	18.5	7.7	8.2	12.9	14.3	15.6
s444	17.9	12.6	16.1	17.6	19.3	10.6	10.6	14.5	17.2	19.0
s499	17.5	16.3	18.6	19.3	20.5	16.3	16.3	18.0	19.3	20.5
s510	17.9	15.9	18.2	19.2	20.1	15.9	15.9	17.4	18.8	20.0
s526	14.1	11.0	12.1	13.5	15.1	9.6	9.6	11.4	13.5	15.1
s526n	14.1	11.0	12.1	13.5	15.1	9.6	9.6	11.4	13.5	15.1
s635	165.9	157.4	113.3	127.9	135.4	4.8	4.9	78.8	97.1	102.6
s641	89.1	66.9	74.9	76.6	78.0	62.6	62.6	66.6	73.3	77.2
s713	90.3	71.9	75.8	77.4	78.7	64.5	64.5	67.5	74.2	78.1
s820	19.7	19.4	20.6	21.6	22.7	19.4	19.4	20.5	21.6	22.7
s832	20.1	19.9	21.1	22.1	23.2	19.9	19.9	21.0	22.1	23.2
s838	25.5	20.8	21.5	23.4	25.0	9.3	10.2	17.0	20.1	22.1
s938	25.5	20.8	21.5	23.4	25.0	9.3	10.2	17.0	20.1	22.1
s953	24.3	21.4	21.7	23.6	25.3	19.4	19.4	21.7	23.6	25.3
s967	21.7	18.6	19.6	21.4	22.8	17.3	17.7	19.6	21.4	22.8
s991	97.5	91.8	65.7	74.2	80.6	79.6	79.6	52.0	62.4	68.1
s1196	21.9	16.2	15.3	18.0	20.2	10.1	8.0	8.2	7.1	8.0
s1238	21.9	16.2	15.3	18.0	20.2	10.1	8.0	8.2	7.1	8.0
s1269	52.3	48.0	35.6	40.6	44.5	43.0	43.0	30.5	39.1	44.0
s1423	93.3	86.6	69.4	73.5	77.3	76.7	76.0	69.2	73.1	75.6
s1488	33.3	30.1	32.6	33.8	35.0	30.1	30.1	32.3	33.7	35.0
s1494	33.9	30.7	33.2	34.3	35.6	30.7	30.7	32.9	34.3	35.6
s1512	40.7	35.9	38.4	40.2	41.6	35.9	35.9	38.4	40.2	41.6
s3271	41.5	30.0	28.4	32.6	35.8	28.8	28.8	15.1	10.0	7.7
s3330	35.9	23.9	26.3	28.9	31.8	18.7	18.7	23.6	25.9	27.6
s3384	86.3	77.6	58.3	65.9	71.7	67.6	67.6	41.6	48.5	52.8
s4863	82.3	75.6	55.6	62.9	68.5	69.2	69.2	44.2	45.4	46.5
s5378	29.5	24.1	25.3	26.0	27.2	23.1	23.1	24.4	25.6	26.7
s6669	129.7	124.8	87.2	98.2	106.4	110.0	110.0	62.5	44.5	49.4
s9234.1	76.9	65.0	65.6	69.8	72.4	55.3	55.3	65.6	69.8	72.4
s9234	76.9	65.0	65.6	69.8	72.4	55.3	55.3	65.6	69.8	72.4
s13207.1	77.1	64.8	63.2	68.9	72.3	54.8	54.8	63.2	68.9	72.3
s13207	86.7	67.6	63.3	69.6	74.0	57.8	57.8	63.2	68.9	72.3
s15850.1	82.3	71.6	67.8	71.9	74.8	58.5	58.5	67.8	71.9	74.8
s15850	117.1	93.0	78.8	88.8	96.3	83.8	83.8	67.8	71.9	74.8
s35932	35.3	35.0	36.4	37.5	37.6	21.1	21.1	26.2	30.0	32.5

Table 14: Clock period improvements of multi-phase ISCAS'89 benchmark circuits.

Circuit	**Improvement TB (%)**				**Improvement CSS (%)**					**Improvement TBCSS (%)**			
Circuit	$I_{1\phi}^{TB}$	$I_{2\phi}^{TB}$	$I_{3\phi}^{TB}$	$I_{4\phi}^{TB}$	$I_{1\phi}^{CSS}$	$I_{2\phi}^{CSS}$	$I_{3\phi}^{CSS}$	$I_{4\phi}^{CSS}$	I_{FF}^{CSS}	$I_{1\phi}^{TBCSS}$	$I_{2\phi}^{TBCSS}$	$I_{3\phi}^{TBCSS}$	$I_{4\phi}^{TBCSS}$
s27	16	-6	-24	-34	20	23	22	18	32	32	18	3	-10
s208.1	33	13	-3	-14	30	17	20	26	57	53	27	18	16
s298	23	22	7	-4	11	6	3	1	32	32	27	9	-3
s344	31	20	13	7	0	0	0	0	31	31	20	13	7
s349	31	20	13	7	0	0	0	0	31	31	20	13	7
s382	28	6	2	-9	16	15	2	1	40	40	20	4	-8
s386	3	-3	-9	-15	0	0	0	0	3	3	-3	-9	-15
s400	28	6	2	-9	16	15	2	1	39	39	20	3	-8
s420.1	27	8	2	-6	36	20	17	15	56	53	26	18	11
s444	30	10	2	-8	16	10	3	1	41	41	19	4	-6
s499	7	-6	-10	-17	0	3	0	0	7	7	-3	-10	-17
s510	11	-2	-7	-12	0	5	2	1	11	11	3	-5	-12
s526	22	14	4	-7	12	6	0	0	32	32	20	4	-7
s526n	22	14	4	-7	12	6	0	0	32	32	20	4	-7
s635	5	32	23	18	97	30	24	24	97	97	53	41	38
s641	25	16	14	13	6	11	4	1	30	30	25	18	13
s713	20	16	14	13	10	11	4	1	29	29	25	18	14
s820	2	-5	-10	-15	0	0	0	0	2	2	-4	-10	-15
s832	1	-5	-10	-15	0	0	0	0	1	1	-4	-10	-15
s838	18	16	8	2	51	21	14	12	64	60	33	21	14
s938	18	16	8	2	51	21	14	12	64	60	33	21	14
s953	12	11	3	-4	9	0	0	0	20	20	11	3	-4
s967	15	10	2	-5	5	0	0	0	20	18	10	2	-5
s991	6	33	24	17	13	21	16	16	18	18	47	36	30
s1196	26	30	18	8	51	47	61	60	54	63	63	68	64
s1238	26	30	18	8	51	47	61	60	54	63	63	68	64
s1269	8	32	22	15	10	14	4	1	18	18	42	25	16
s1423	7	26	21	17	12	0	0	2	18	19	26	22	19
s1488	10	2	-1	-5	0	1	0	0	10	10	3	-1	-5
s1494	9	2	-1	-5	0	1	0	0	9	9	3	-1	-5
s1512	12	6	1	-2	0	0	0	0	12	12	6	1	-2
s3271	28	32	22	14	4	47	69	79	31	31	64	76	82
s3330	33	27	19	11	22	10	10	13	48	48	34	28	23
s3384	10	32	24	17	13	29	26	26	22	22	52	44	39
s4863	8	32	24	17	8	21	28	32	16	16	46	45	43
s5378	18	14	12	8	4	3	2	2	22	22	17	13	9
s6669	4	33	24	18	12	28	55	54	15	15	52	66	62
s9234.1	15	15	9	6	15	0	0	0	28	28	15	9	6
s9234	15	15	9	6	15	0	0	0	28	28	15	9	6
s13207.1	16	18	11	6	15	0	0	0	29	29	18	11	6
s13207	22	27	20	15	15	0	1	2	33	33	27	21	17
s15850.1	13	18	13	9	18	0	0	0	29	29	18	13	9
s15850	21	33	24	18	10	14	19	22	28	28	42	39	36
s35932	1	-3	-6	-6	40	28	20	14	40	40	26	15	8
Average	16.7	15.3	8.0	1.6	16.5	12.1	11.4	11.3	30.3	30.3	24.8	17.7	12.0

Table 15: Circuit info and run times for multi-phase ISCAS'89 benchmark circuits.

Circuit Info			**Time (sec)**							
Circuit	r	p	$t_{1\phi}^{TB}$	$t_{1\phi}^{TBCSS}$	$t_{2\phi}^{TB}$	$t_{2\phi}^{TBCSS}$	$t_{3\phi}^{TB}$	$t_{3\phi}^{TBCSS}$	$t_{4\phi}^{TB}$	$t_{4\phi}^{TBCSS}$
s27	3	4	0	0	0	0	0	0	0	0
s208.1	8	28	0	0	0	0	0	0	0	0
s298	14	54	0	0	0	0	0	0	0	0
s344	15	68	0	0	0	0	0	0	0	0
s349	15	68	0	0	0	0	0	0	0	0
s382	21	113	0	0	0	0	0	0	0	0
s386	6	15	0	0	0	0	0	0	0	0
s400	21	113	0	0	0	0	0	0	0	0
s420.1	16	120	0	0	0	0	0	0	0	0
s444	16	113	0	0	0	0	0	0	0	0
s499	22	462	0	0	0	0	0	0	1	1
s510	6	15	0	0	0	0	0	0	0	0
s526	21	117	0	0	0	0	0	0	0	0
s526n	21	117	0	0	0	0	0	0	0	0
s635	32	496	0	0	0	1	0	1	1	1
s641	19	81	0	0	0	0	0	0	0	0
s713	19	81	0	0	0	0	0	0	0	0
s820	5	10	0	0	0	0	0	0	0	0
s832	5	10	0	0	0	0	0	0	0	0
s838	32	496	0	0	0	0	0	1	1	0
s938	32	496	0	0	0	0	0	1	1	0
s953	29	135	0	0	0	0	1	0	0	0
s967	29	135	0	0	0	0	1	0	0	0
s991	19	51	0	0	0	0	0	0	0	0
s1196	18	20	0	0	0	0	0	0	0	0
s1238	18	20	0	0	0	0	0	0	0	0
s1269	37	1260	0	0	0	0	0	0	1	1
s1423	74	1471	1	1	1	2	2	2	2	3
s1488	6	15	0	0	0	0	0	0	0	0
s1494	6	15	0	0	0	0	0	0	0	0
s1512	57	415	0	0	0	1	1	1	1	1
s3271	116	789	1	1	1	1	1	1	2	2
s3330	132	514	0	0	1	1	1	1	2	2
s3384	183	1759	1	1	2	2	3	3	4	4
s4863	104	620	0	1	1	1	1	1	1	2
s5378	179	1147	1	1	1	2	2	2	3	3
s6669	239	2138	1	2	2	3	3	4	4	6
s9234.1	228	247	2	3	2	3	4	5	5	6
s9234	211	2342	2	3	3	4	4	5	5	7
s13207.1	669	3068	3	4	6	7	9	11	13	15
s13207	669	3068	3	5	6	8	9	13	13	19
s15850.1	534	10830	10	25	13	30	19	37	25	47
s15850	597	14257	15	26	19	32	24	42	30	46
s35932	1728	4187	6	8	16	17	21	28	27	39

7.0 APPLICATIONS TO RESONANT CLOCKING

Development of a low-jitter, low-skew (or controllable skew for clock skew scheduling) clocking technology that has low power dissipation is one of the major research topics in the development of next-generation synchronous integrated systems. Among the proposed clocking technologies are wireless [22, 23, 47] and transmission line-based [4, 10, 11, 13, 17, 28, 54, 55, 62, 96, 97] approaches. These technologies must be supported by specific design flows and CAD suites in order to be viable in semiconductor implementation. In this chapter, the adaptability of design and analysis methods presented in Chapters 4, 5 and 6 to the physical design flow of circuits synchronized by a transmission line-based clocking technology is described. The particular transmission line-based technology of interest, the rotary clocking technology, is described in detail.

Resonant clocking technologies are described in Section 7.1. The operation of rotary clocking technology published in [93, 94, 96, 97] is summarized in Section 7.2. The timing of circuits synchronized with the rotary clocking technology is discussed in Section 7.3. The chapter is summarized in Section 7.4.

7.1 RESONANT CLOCKING

In the last decade, the frequencies of the clock signals for the state-of-the-art digital integrated circuits have surpassed the GHz milestone [63]. Historically, systems that operate at clock frequencies in low MHz ranges have utilized off-chip quartz crystal oscillators [38, 59]. The oscillatory signal generated off-chip with the quartz crystal is input to the on-chip PLL, where it is multiplied to the desired frequency on chip. The generated signal is distributed

to the synchronous components throughout the chip, typically using a tree topology, called a *clock tree network* [26]. Especially in nano-scale CMOS, where signal integrity has become a dominating problem, the distribution of the clock signal from a single clock source over a clock tree network has become quite error-prone. The discrepancies in the arrival time of the clock signal at the destination registers increase with each new technology. Also, the on-chip PLL components occupy chip area and lead to problems with signal reflections, capacitive loading and power dissipation that effectively limit the maximum operating frequency.

For high MHz ranges or GHz frequencies, the generation of the clock signal with an off-chip oscillator has become a tedious task. The prevailing methodology to generate such high-frequency clock signals is to use on-chip frequency multiplication by using phase-locked-loop (PLL) components [14].

The resonant clocking technologies [4, 10, 11, 13, 17, 28, 54, 55, 62, 96, 97] present an alternative to generating the synchronizing clock signal. The resonant clocking technologies eliminate the necessity to use a complicated on-chip PLL component. The implementation of resonant clocking technologies require long interconnects on the chip, which are modeled by transmission lines. Instead of the lossy RC characteristics of long wires, LC characteristics of transmission lines provide the physical medium for oscillation. A common signal is excited and kept oscillating on the transmission lines, which constitutes the global clock signal.

Currently, there are three major types of resonant clocking technologies. These resonant clocking technologies are categorized with respect to their oscillator types:

1. Coupled LC oscillator based resonant clocking technology [10, 11, 62],

2. Standing wave oscillator based resonant clocking technology [4, 13, 54, 55],

3. Traveling wave oscillator based resonant clocking technology [93, 94, 96, 97].

Coupled LC oscillator based resonant clocking technology provides a constant magnitude clock signal with constant phase. A clock signal with constant magnitude and constant phase is similar to the conventional clock signals that are delivered using conventional clock tree networks. The main advantage of coupled LC oscillator based resonant clocking technology

Table 16: Categorization of the resonant clocking technologies.

Oscillator Type	Phase	Voltage
Coupled LC	Constant	Constant
Standing Wave	Constant	Variable
Traveling Wave	Variable	Constant

over other resonant clocking technologies is that coupled LC oscillator based clocking provides the desired clock signal without any change to the conventional design flows. Higher circuit performances are achievable solely by replacing the clock distribution network with the coupled LC oscillator based resonant clocking technology distribution network.

Standing wave oscillator based resonant clocking technology provides a varying amplitude clock signal with a constant phase. Similar to coupled LC oscillator based resonant clocking technology, clock phase is constant, thus this technology does not require drastic modifications to the conventional design flows.

Traveling wave oscillator based resonant clocking technology is the resonant clocking technology of interest in this dissertation. Traveling wave oscillator based resonant clocking technology, also called *rotary clocking technology*, provides a clock signal which has a constant magnitude and varying phase. Varying phase (delay) of clock signal provides permits easy implementation of non-zero clock skew systems. The design and analysis methods proposed for non-zero clock skew systems in Chapters 4, 5 and 6 can be used to design circuits synchronized with the rotary clocking technology.

Table 16 [61] summarizes the categorization of the presented resonant clocking technologies, based on the magnitude and phase properties of the generated clock signals.

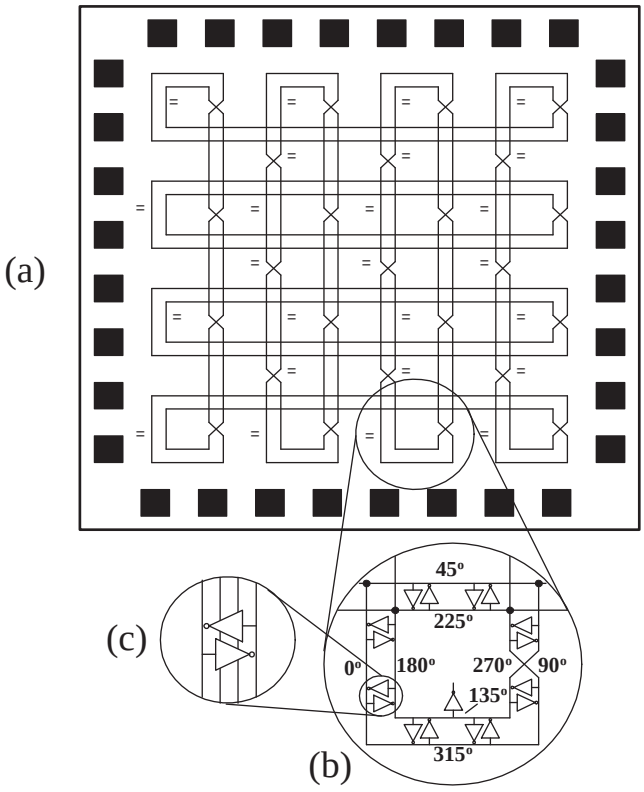


Figure 46: Basic rotary clock architecture.

7.2 ROTARY TRAVELING WAVE OSCILLATORS

Rotary traveling-wave oscillators (RTWO's) comprise a novel clock network implementation technology providing controllable-skew, low-jitter, gigahertz range clocking with fast transition times and low power consumption [96]. The networks formed by RTWO's are scalable to any practical circuit dimension. RTWO's are generated on cross-connected transmission lines, constructing a differential LC transmission line oscillator. These oscillators generate multi phase (360 degrees) square waves with low jitter. Multiple RTWO's can be connected together forming the rotary oscillator arrays (ROA) which distribute the synchronized square wave over the whole chip. The basic ROA structure [96] is shown in Figure 46. This arrangement produces a clock signal in each ring which sweeps around the ring in a frequency

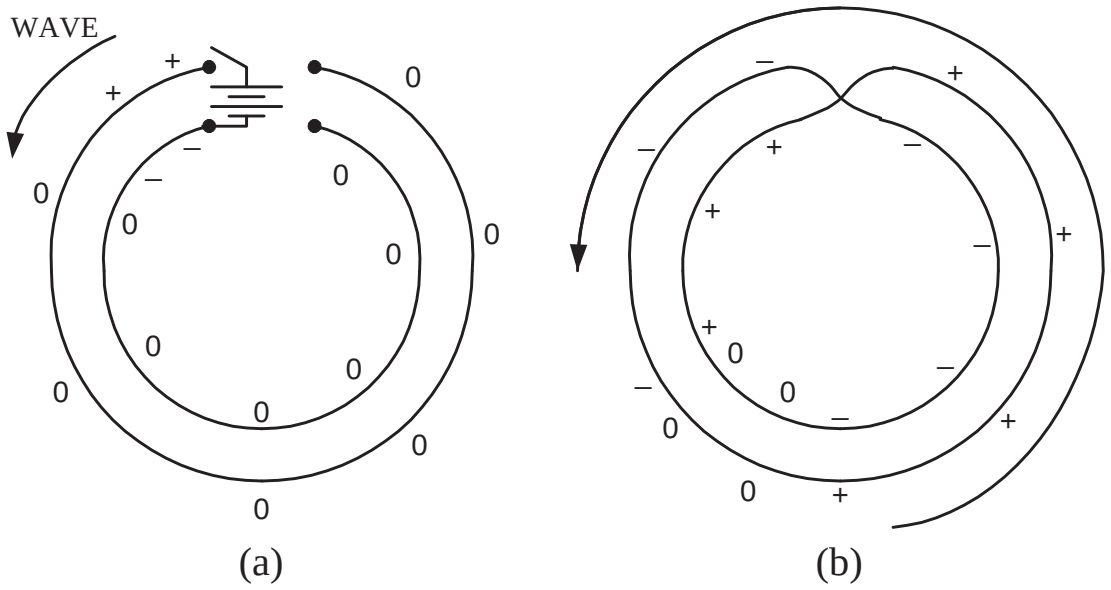


Figure 47: The RTWO theory.

dependent on the electrical length of the ring. Pulses on each ring are phase-locked via the shared transmission line wires between the rings.

The current paths along the cross-connected transmission lines are terminated to each other causing the switching energy to recirculate within the system as transmission line energy. The energy circulation provides significant power savings over conventional, buffered clock tree networks. The synchronization capability is enhanced, providing support for both single and multi-phase clocking schemes. The frequency of the clock signal generated by RTWO's is limited only by the cutoff frequency of the integrated circuit technology used, and can be manipulated by changing the length or adjusting the loading impedances of the RTWO structures (ROA rings).

Figure 47 illustrates the operation of an individual RTWO [96]. Figure 47(a) shows the open loop that conceptually occurs when the circuit is being excited for the first time. Figure 47(b) shows the closed loop in steady state of operation where overlap of the traveling waves causes signal negation.

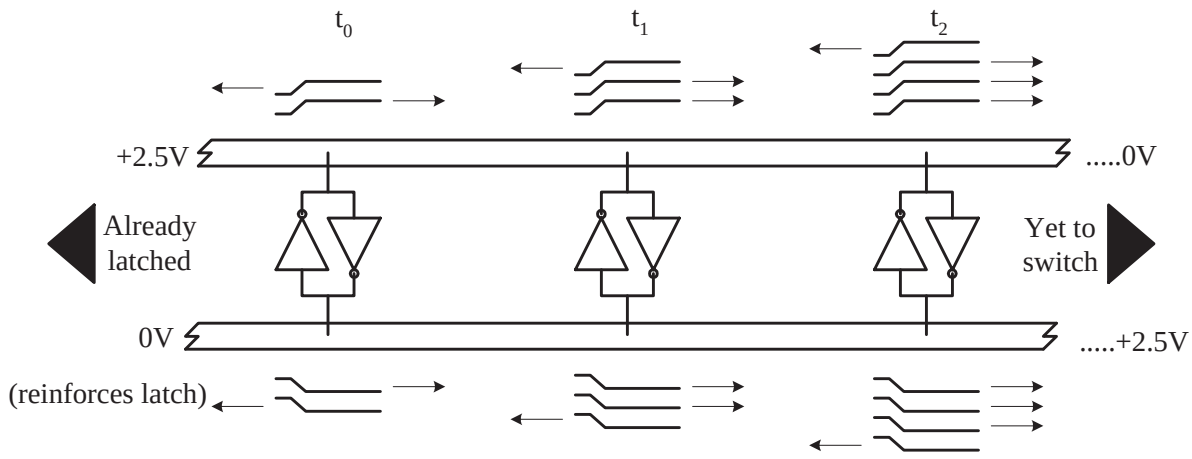


Figure 48: The cross-section of the transmission line with shunt connected inverters.

When the transmission line is excited from one or more points, the traveling wave is established on the cross-connected line. The traveling wave is inverted on the crossover points, generating different phases of the square wave. Any odd number of crossovers are allowed on the transmission line, where the number of crossovers determine the number of clock phases for the multi-phase clock signal generated by the rotary clocking technology. The duty cycles of the multiple clock phases are determined by the location of the crossovers on the ring. Also, the relative phase and skew of any point on the ring is well known due to the homogeneity of the traveling pattern around the ring. Note that anti-parallel (shunt connected) inverter pairs are used between the cross-connected lines to save power, initiate and maintain the traveling wave. After excitation, the anti-parallel inverters feed the traveling wave in the stronger direction, up to a stable oscillation frequency. The transmission line with anti-parallel connected inverters is shown in Figure 48 [96]. In Figure 48, the traveling wave is traveling from left to right.

Each pair of anti-parallel inverters on the path of the traveling signal turns on after some time, stimulating the same process at the neighboring pair of anti-parallel inverters in the direction of the wave. The transmission line impedance is on the order of 10Ω and the

differential on-resistance of the anti-parallel connected inverters are in the 100Ω-1kΩ range for a 0.25um technology [96].

Once a wave is established, it takes little power to sustain it. The dissipated power on the ring is given by the I^2R dissipation instead of the conventional CV^2f expression. Such consideration of power is possible because the energy that goes into charging and discharging MOS gate capacitance (of the inverters) becomes transmission line energy, which in turn is circulated in the closed electromagnetic path. Such conservation of energy is enabled by adiabatic switching [16, 40], in terminating the current path to the transmission line, instead of ground. The coherent switching occurs only in the direction of the traveling path. An equal amount of energy is launched in the reverse direction, however the latches in this direction are already switched, thus this energy simply serves to reinforce the previous switching events on these registers.

The frequency of the clock signal generated by the rotary clocking technology depends on the length of the rotary wires [96]. On a typical RTWO loop, the oscillation frequency of the signal is given by the equation:

$$f_{osc} = \frac{1}{2\sqrt{L_{total}C_{total}}} \tag{7.1}$$

L_{total} is the total loop inductance and is proportional to the length of the ring. L_{total} also depends on the width of the wire that builds the ring, providing a design flexibility to generate the desired frequency. Inductance variation on a typical silicon implementation is expected to be small, because of the high quality of lithographic reproduction [96]. C_{total} is the total capacitance that is driven by the RTWO ring. The majority of the driven capacitance is the gate-oxide capacitance of the drive FETs and clock load FETs. Post-fabrication, the projected variation in the targeted operating frequency is 5%, accounting for the sources of variation and the dependence of the operating frequency on $\sqrt{C}$ and $\sqrt{L}$ [96].

A detailed analysis of the rotary clocking technology and the RTWO loops can be found in [93, 94, 96, 97]. The point of interest in the work is mainly the timing requirements of the rotary clocking technology, which are presented in Section 7.3.

7.3 TIMING REQUIREMENTS OF ROTARY CIRCUITS

As described in Section 7.2, rotary clocking technology provides a constant magnitude clock signal with varying phase (clock skew and clock phase). The constant magnitude of the clock signal is similar to the customary, however, varying clock phase is not popular in mainstream circuit design flow. It is more common to use a zero clock skew, single-phase clock signal in synchronous circuit design due to its simplicity in design and analysis. The majority of the design automation tools for clock tree synthesis produce better results in generating a clock distribution network that provides a zero clock skew, single phase clock distribution as opposed to a non-zero clock skew tree. Non-zero clock skew and multi-phase synchronization, although shown in Chapter 6 to be consistently superior over traditional zero clock skew, single-phase design, is not very popular due to the lack of automation.

The characteristics of the clock signal generated by the rotary clocking technology are investigated for the applicability of the advanced timing and synchronization methodologies presented in the dissertation. It is shown that, the synchronization capabilities of the rotary clocking technology are fully-compatible with the non-zero clock skew, multi-phase circuit design methodology. Level-sensitive design techniques can also be flawlessly integrated into the design flow.

Unlike traditional PLL-based clock sources, the generation of a multi-phase clock signal is highly practical with rotary clocking. The number of phases in the clock signal generated by the rotary clocking technology is determined by the number of cross-overs in the ROA rings. It is reported [95] that two, four and eight phase synchronization schemes can be implemented with rotary clocking technology, without loss of quality.

For traditional PLL-based clock sources and clock tree networks, excessive amount of buffering can be necessary in order to deliver the clock signals to the synchronous component with the desired delays. Remember from Section 5.4 that buffer elements are available only in discrete values. In traditional clock tree networks, clock delays are generated with buffering, thus, clock delays are available only in discrete values for such systems. For rotary clocking technology, however, buffer elements are not necessary, as clock delays are provided with the propagation of the clock signal on ROA rings. The clock phase driving a synchronous

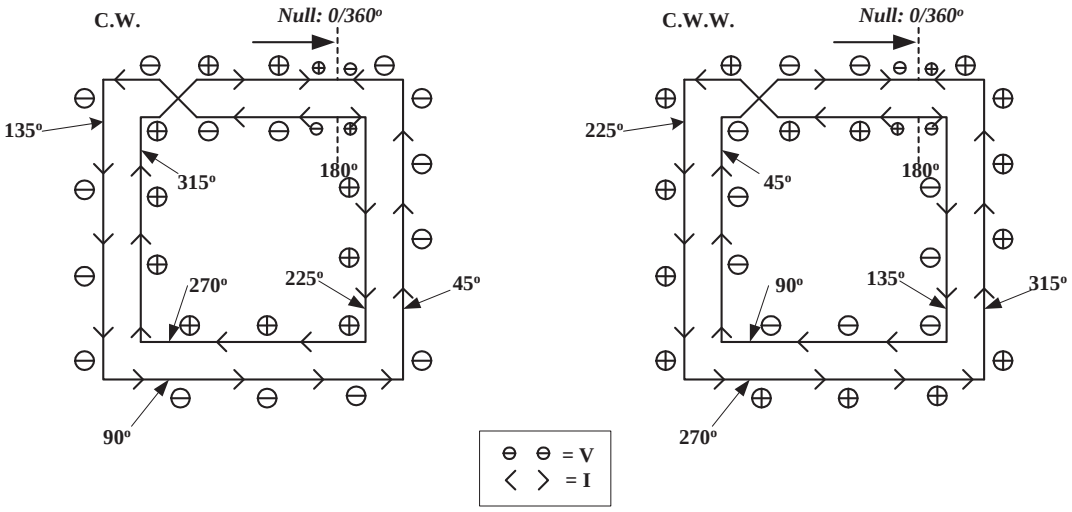


Figure 49: The clock phase relationships on an ROA ring.

component is determined by the location of the connection point of the clock signal wire on the ROA ring as shown in Figure 46(b) (page 116). Figure 49 also presents the different phases of the clock signal available for a sample rotary implementation with one cross-over point. Note that with this implementation, two corresponding points on the differential line provides clock signals with are shifted by 180 degrees in time.

Rotary clocking technology readily supplies a fine grain of clock delays (phases). From a CAD perspective, continuous delay models can be used to model clock delays available in the network. From a circuit design perspective, the assignment of different clock delays to the synchronous components of a rotary-clock synchronized circuit are essential for the proper operation of the circuit. If a circuit previously designed to operate with zero clock skew is synchronized with rotary clocking technology, synchronization problems may occur. The most common problem is due to an unbalanced loading of the clock network. In order to preserve the synchronization of the original zero clock skew circuit, all synchronous components must be driven by the same location on the ROA ring. Such a load distribution may affect the rotation of the oscillatory signal on the ring, thereby causing degradations in the quality of synchronization. Also, due to simultaneous switching of synchronous com-

ponents, this type of distribution might lead to thermal hot spots on the chip area. In the optimal scheduling scenario, the clock delays at the synchronous components are distributed relatively evenly in time, leading to a relatively balanced distribution of the latching points on the rotary ring. The required balanced loading of the ROA rings can be provided by clock skew scheduling (see the distribution of clock delays for a sample circuit in Figure 25 on page 63).

The advanced timing methodology of using non-zero clock skew circuits with multi-phase synchronization can easily be realized in circuits synchronized with rotary clocking technology. Advantageously, implementation of circuits synchronized with the rotary clocking technology mutually requires the automated design and analysis methodologies for multi-phase, non-zero clock skew synchronization schemes. Such integration of the design and analysis methodologies into the physical design flow leads to circuits which benefit both from the presented advanced timing methodologies and the rotary clocking technology.

7.4 SUMMARY

In this chapter, a family of next-generation clocking technologies, resonant clocking technologies, is reviewed. It is shown that resonant rotary clocking technology inherently supports multi-phase and non-zero clock skew synchronization. The integration of the design and analysis methods presented in this dissertation to the physical design flow of circuits synchronized with the rotary clocking technology are proposed. This integration is described in detail in Chapter 8.

8.0 PHYSICAL DESIGN USING RESONANT CLOCKING

This chapter describes a physical design methodology for circuits synchronized with a resonant clocking technology (a general discussion of resonant clocking technologies is presented in Chapter 7). In particular, synchronization with the resonant *rotary* clocking technology [93, 94, 96, 97] is analyzed. Rotary clocking technology provides a controllable-skew, low-jitter clock signal with fast transition times and low power consumption. As discussed in Section 7.3, the availability of the full spectrum of clock phases (delays) promotes rotary clocking technology as an excellent candidate for the synchronization of high-performance, non-zero clock skew circuits.

The physical design methodology for rotary clocking technology is presented in order to demonstrate the practical application of the advanced timing and synchronization methodologies discussed in this dissertation. The effects of nano-scale CMOS design to this application are considered at various steps of the described physical design methodology.

The three major steps of the presented physical design methodology are the *partitioning*, *clock skew scheduling* and *placement* steps. The partitioning step is proposed in order to generate logic partitions that are implementable within the ROA ring regions of a rotary clocking network. The clock skew scheduling step is required to improve the scalability of conventional clock skew scheduling techniques by benefiting from the results of partitioning. The placement step is proposed in order to provide a practical implementation alternative for the mapping of the circuit logic and registers to the ROA rings. These steps are required to increase the feasibility of the rotary clocking technology as the infrastructure of choice for the advanced timing methodologies discussed in previous chapters (such as clock skew scheduling and multi-phase synchronization).

This chapter is organized as follows. In Section 8.1, an outline of the physical design flow is presented. In Section 8.2, the CAD implementation of the physical design flow is described in detail. In Section 8.3, experimental results of various stages of the design methodology are presented. A summary is offered in Section 8.4.

8.1 PHYSICAL DESIGN FLOW

Synchronization of digital VLSI circuits with the rotary clocking technology and the integration of clock skew scheduling into the circuit design flow require methodical introduction. In this section, these new design paradigms are outlined from the physical design and electronic design automation points of view.

The design flow is illustrated with the flow chart shown in Figure 50. The flow includes processing the design entry to investigate the complexity and requirements of the circuit, partitioning the netlist, performing clock skew scheduling and performing register and logic placement.

The design entry is provided in industry standard file formats, such as DEF, LEF and SDF file formats [72]. An initial timing information of the circuit is necessary for the application of clock skew scheduling. This information can be obtained by performing static timing analysis to a preliminary placement and routing or from a silicon virtual prototype [9] of the circuit.

The implementation of the ROA rings and netlist partitioning are dependent on each other as illustrated in the *Partitioning* step in the flow chart. The size and number of rings in the ROA structures depend on several factors such as the complexity of the design, the availability of clock network design resources, the computational resources for timing analysis, and the availability of silicon area. Despite these dependencies, the number and dimensions of ROA rings in a circuit are quite flexible. The number of ROA rings is usually held sufficiently high in order to limit the total wire length. The shapes of ROA rings are not necessarily regular (*e.g.*, rectangles) as implied by the mesh structure presented

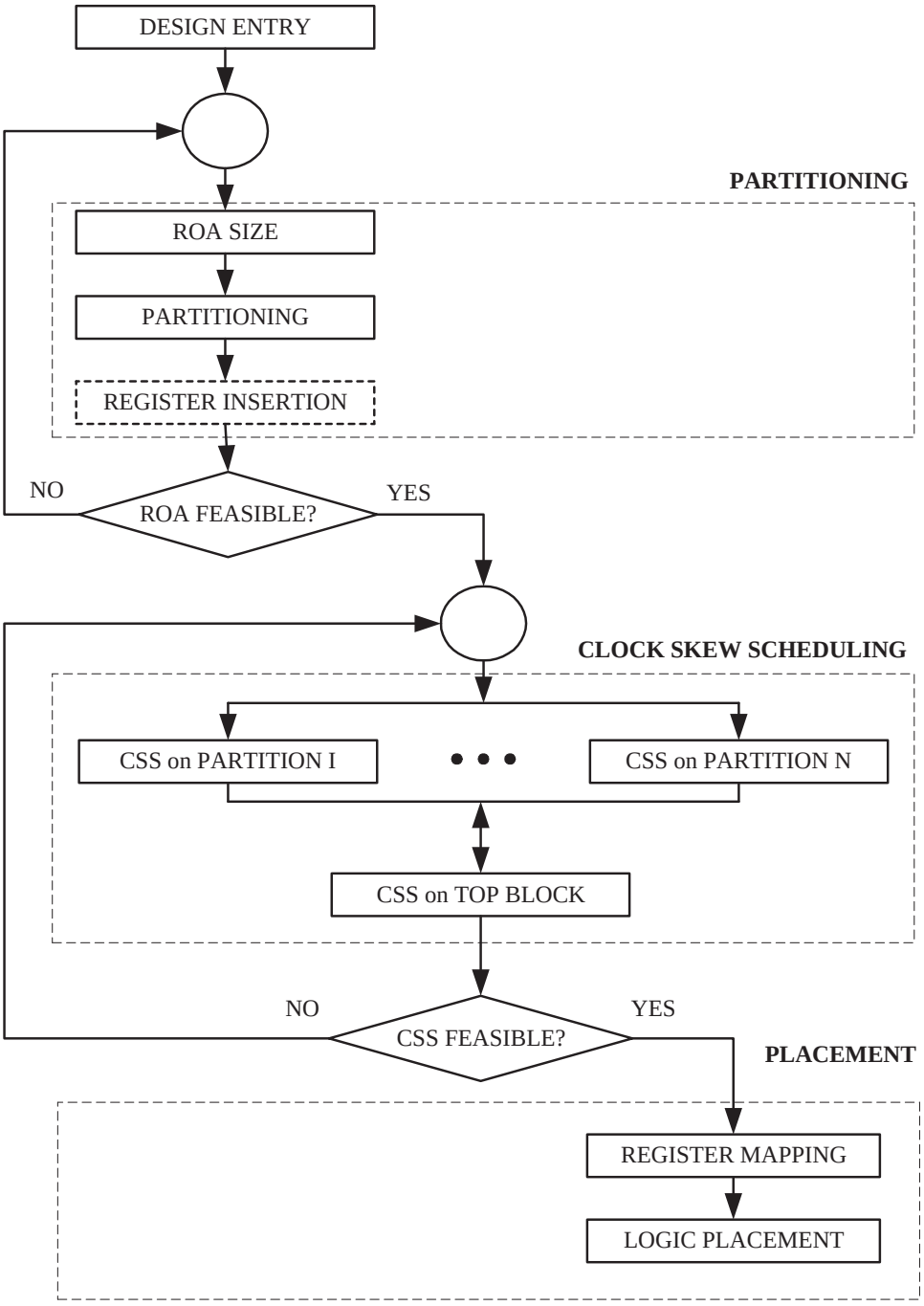


Figure 50: The physical design flow of VLSI circuits with RTWO clock synchronization.

in Section 7.2. Such flexibility in the physical implementation of the ROA rings enables reconciliation of the non-routable blocks of the chip area.

Partitioning is performed on a gate-level or a register-transfer level netlist. For the former case, it is often necessary to insert extra registers in the logic network as part of the timing-driven partitioning process. This process is represented by the "Register Insertion" block in the flow chart. These inserted registers are level-sensitive latches operating in the transparent phases of operation (Chapter 3) in order to preserve the functionality of the original circuit. The feasibility of the partitioning result is checked at the next validation step. If the current result is not feasible, the *partitioning* step of the design flow is repeated until feasibility is satisfied.

In the *clock skew scheduling* step, the rotary clock network is constructed. Data paths that are local to each partition are identified and the corresponding timing constraints are included in the clock skew scheduling problem for that partition. Similarly, the timing constraints of local data paths which span different partitions are included in the clock skew scheduling problem of the top block. A heuristic method is proposed to solve the partition and top block LP problems. The clock skew scheduling problems of each partition are independent of each other, so these analyses can be parallelized.

After the *clock skew scheduling* block is completed, the optimal clock signal delays required at each synchronous component are known. Depending on the number of clock phases and the number of registers for a given clock phase, the mapping of synchronous components to the registers within an ROA ring is performed. This is an automated design step called "Register Mapping" in the flow chart. Following register mapping, the rest of the logic within a partition is placed in the area available within the ROA rings for this partition. This placement is performed using conventional logic placement techniques.

The partitioning and clock skew scheduling steps of the physical design flow are presented in detail in Sections 8.1.1 through 8.1.4. The placement step is discussed in Section 8.1.5.

8.1.1 Timing-Driven Partitioning

Timing-driven partitioning refers to the partitioning process during the placement and routing steps of conventional physical design flow of integrated circuits. The objective of the conventional timing-driven partitioning process is to generate circuit placements that are more likely to meet a particular timing budget. Path-based and net-based partitioners [1] are the two most widely used kinds of partitioners in current state-of-the-art physical design. Both path-based and net-based partitioners are used to limit the lengths of selected critical paths in a circuit. Such limitation in the number of analyzed paths significantly reduces the processing time for partitioning (and static timing analysis) while generally preserving the accuracy of the analyses.

In clock skew scheduling, the local data paths in an entire circuit (or circuit partition) are equally important and analyzed together. Thus, an alternative partitioning approach is proposed in this work using selection criteria that lead to partitions which are amenable to clock skew scheduling. Traditional path-based and net-based timing-driven partitioning methods are not used. Instead, a hypergraph partitioning tool is used to generate partitions that are amenable to clock skew scheduling and easily implementable with the rotary clocking technology. Principally, timing-driven partitioning is performed within the proposed design methodology subject to the following considerations:

1. To construct the logic network partitions that will be synchronized by individual ROA rings of the rotary clocking technology.
2. To enable the completion of path enumeration on large scale circuits.
3. To enable the completion of clock skew scheduling algorithms on large scale circuits.

The first of the three factors listed above is directly related to the implementation of the rotary clocking technology. If clock tree synthesis is performed completely independent from logic synthesis, the assignment of synchronous components to individual ROA rings can be inefficient for physical implementation. As discussed in Section 7.3, a relatively balanced distribution of clock phases is necessary for the quality of synchronization with a rotary clock signal. An unbalanced loading of synchronous components to ROA rings may also cause hot spots in the circuit or significantly increase the clock load on one side of the chip compared to

another (thereby causing performance degradation). To prevent such negative effects where a fraction of the registers must be connected to ROA rings outside the close proximity of the register location, logic and clock tree synthesis need to be performed interdependently. The partitioning procedure presented here achieves this goal by generating balanced logic partitions to be synchronized by each ROA ring. Advantageously, the clock phases at the synchronous components within each partition are well distributed after the application of clock skew scheduling (see Figure 25 on page 63) to the logic partitions.

The second and third factors that drive the timing-driven partitioning process are related to the design and analysis methodologies of large-scale circuits. Although discussed here within the context of rotary clock synchronization, the partitioning procedures presented in this dissertation can also be applied to circuits synchronized with traditional clocking technologies. From a CAD perspective, the generality of the partitioning procedure to improving the scalability of clock skew scheduling (independent of the particular clocking technology) is discussed next.

As reported earlier in Chapter 4, scalability of clock skew scheduling is an important criteria for its widespread acceptance in mainstream design. Most industrial-strength timing tools or circuit designers that implement variations of clock skew scheduling perform these tasks only on certain portions of the circuit, without analyzing the circuit in its entirety. Analysis of the entire circuit in order to implement a full-scale application of clock skew scheduling can be computationally intensive for very large-scale circuits. Two main obstacles for the application of clock skew scheduling to the entire circuit are *path enumeration* and *run times of LP model problems*.

With increased logic depth and complexity in state-of-the-art integrated circuits, path enumeration becomes highly costly and intractable. In practice, hierarchical timing models [71] are used in order to simplify the enumeration of paths in the circuit. In a flattened circuit, however, path enumeration can not always be completed within reasonable time and computation resources. Partitioning, as proposed in this work, remedies this shortcoming. Generally, very long paths are split with a cut and a level-sensitive latch in the transparent phase is inserted on the cut. The transparent phase latch has no effect on the functionality

of the circuit because the data signal immediately propagates through the latch. This latch, however, simplifies path enumeration by shortening the logic depth of the original path.

Once path enumeration is complete, the LP problem for the application of clock skew scheduling is formulated as described in Chapter 4. The LP problems generated for an integrated circuit with millions of paths and hundreds of thousands or more synchronous components can be very large. The run times of such large LP problems are usually reasonable within the typically long IC design cycles (up to a few days with industrial strength LP solvers and common computing resources). However, very large models might not be solvable at all within the memory limits of common computing resources. In several industry applications, for instance, LP model problems for the clock skew scheduling of large-scale circuits are observed to exceed the practical limits of standard industrial strength computing resources (e.g. 4 gigabytes of memory for 32-bit systems) [50].

8.1.2 Partitioning with Chaco

In the development of the partitioning step of the physical design flow, the partitioning tool *Chaco* [31] from Sandia National Laboratories is used. Chaco is a hypergraph partitioning tool that is primarily developed for the parallelization of tasks on special architectures. Nevertheless, Chaco has been proven to be applicable to a wide range of areas. Chaco offers various methods (spectral bisectioning [57], the inertial method [91], the Kernighan-Lin [39], Fiduccia-Mattheyses [20] algorithms and multilevel partitioners [32]) for partitioning, each fine tuned for a specific purpose.

Among the multiple criteria for partitioning a synchronous circuit for clock skew scheduling are the weight, number and location of the cuts amongst partitions, the weight of each partition, the relative mapping of sequentially-adjacent registers to partitions and the number of internal vertices per partition. Chaco tracks the quality of these partitioning performance metrics with user-defined priorities. In order to generate partitions amenable to clock skew scheduling, the number of cuts between partitions must be minimal and the number of internal vertices (vertices that do not have edges between partitions) must be maximal.

Depending on particular design budgets, the priority of the performance metrics, or the weights of particular nets or vertices can be fine tuned.

In the computer-aided design (CAD) tool implementation, the application of partitioning to two types of netlists are supported. These netlists, categorized by the hierarchical level of input data, are:

1. Gate level netlists,
2. Register-transfer level netlists.

If the input to the CAD tool is a register-transfer level netlist, identifying local data paths (register-to-register timing paths) is inherently simple. The local data paths in the register-transfer level netlist already form a circuit graph such as the one shown in Figure 4 (page 13), where each vertex is a register or a synchronous component and each edge is a local data path. If the input to the CAD tool is a gate-level netlist, one of two methods is applied. If the input is small enough such that explicit path enumeration can be performed to generate the circuit graph, the enumeration is performed. For large inputs, where explicit path enumeration cannot be completed, the partitioner is tuned such that registered-input, registered-output partitions are generated. To encourage the generation of such partitions, the following rules are applied in weight assignment to edges:

1. If the edge is between two registers, assign low edge weight.
2. If the edge is a fanout from the data output terminal of a synchronous component to a combinational component, assign high edge weight.
3. If the edge is a fanout from a combinational component to the data input terminal of a synchronous component, assign low edge weight.
4. If the edge is between two combinational components, assign high edge weight.

The Chaco partitioning tool minimizes the weight cuts, leading the cuts to pass through the data inputs terminals of synchronous components. In case of single input synchronous components, like flip-flops, a data input net is singlefold, while a data output net can have multiple fanouts. Hence, the cuts are directed to occur at the data input terminal of a synchronous component as opposed to a data output terminal. A synchronous component on the boundary of two partitions is shared between two partitions, structuring the registered-

input and registered-output partitions. The enforcement of the edge weights only on data I/O terminals (as opposed to all terminals) are to avoid forcing artificial constraints on irrelevant I/O terminals, such as synchronization and scan-path I/O terminals.

The Chaco partitioning tool is operated with different priorities assigned to the partitioning objectives. Experimentally, a *balanced* priority assignment between minimizing the total cut weight and maximizing the number of internal vertices was found to be sufficiently effective.

8.1.3 Register Insertion for Partitioning

As discussed in Section 8.1.2, partitioning can be performed on netlists at two different hierarchy levels. The application of partitioning on a register-transfer level netlist is simpler compared to its application on a gate-level netlist. On a partitioned register-transfer level netlist, a cut is assumed to pass through an arbitrary location on the cut local data path. The final register of the cut local data path is called a *boundary register*. Timing constraints for the local data paths, where the boundary register is either the initial or the final register of the path and the other register is within the same partition with the boundary register are grouped into a *partition LP* problem. Timing constraints of the local data paths between the boundary register and registers in other partitions are grouped into the *top block LP* problem.

When a gate-level netlist is used, the heuristic described in Section 8.1.2 is used to bolster cuts on the input of synchronous components. Unlike its treatment for a register-transfer netlist, a final register of the local data path must be in the same partition with the cut local data path. This objective suggests registered-input, registered-output partitions, simplifying the timing analysis. The slight variation in the weight (or load) balance of the partitions is insignificant and eventually balances out as the transfer of registers between partitions occurs in all directions.

For instances where the partitioner validates a cut on a net that is between two combinational components, *register insertion* is used to satisfy the registered-input, registered-output scheme. The number of inserted registers depends on the quality of the partitioner and the

complexity of the design. In the performed experiments, the number of inserted registers has been observed to be directly proportional to the number of partitions. For higher number of total partitions, the number of inserted registers can get even higher than the number of original registers. This requires the partitioning step to be applied with caution in designs where die area is a scarce resource.

The registers inserted into the logic network in the register insertion step of the physical design flow can affect the functionality of the circuit. In order to preserve the functionality of the circuit, level-sensitive latches are used. The inserted registers are selected as level-sensitive latches operating in their transparent phases of operation (Chapter 3). The propagation of the data signals on the inter-partition paths are not disrupted, as these signals are immediately propagated through the level-sensitive latches during the transparent phases. Constraints similar to the linearized timing constraints presented in Chapter 4 are used in this step in order to drive the inserted registers with proper clock phases (delays).

The general partitioning process is illustrated in Figure 51. In this figure, the black dots represent registers and the lines represent the data paths. The paths from partition (4,1) are demonstrated. Note that only some of the registers and paths are shown. The data paths which are on a *cut* are identified and the timing constraints of these paths are included within the top block LP.

8.1.4 Clock Skew Scheduling of Partitions

In this section, the application of clock skew scheduling on the partitions generated by the timing-driven partitioner is described. A heuristic method is presented in order to perform the referred application. It is shown that this heuristic method, although simplifying clock skew scheduling, is not guaranteed to reach an optimal solution. The heuristic method is described explicitly for circuits synchronized by the rotary clock technology in this chapter, however, it can be generally applied to any synchronous circuit.

The heuristic method to solve the clock skew scheduling of partitions is as follows. Assume that there are n partitions. The partition LP problems ($LP_1, LP_2, \ldots LP_n$) are generated for these n circuit partitions. Each partition LP_i is solved (sequentially or in parallel)

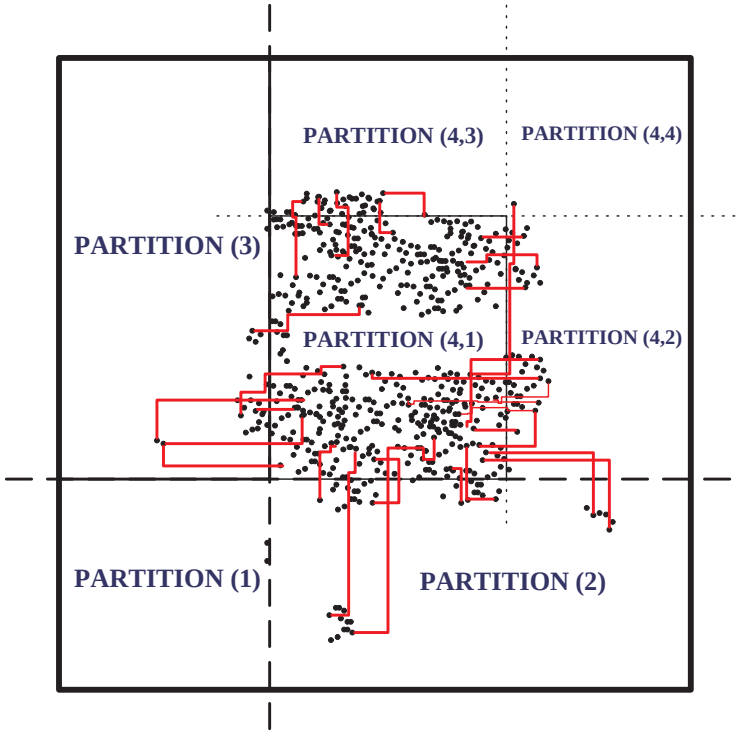


Figure 51: Partitioning a circuit for timing analysis.

in order to compute the minimum clock period permitted by that partition. For proper operation of the circuit, all partitions must operate at the same clock period. The partitions can freely operate at any clock period higher than the minimum clock periods computed for their partition LP problems. Thus, the maximum of the minimum clock periods reported from each partition LP is selected as the principal clock period at which all the partitions are operable. This maximum value corresponds to the frequency at which at least one of the partitions is operating at its maximum frequency, while the rest of the partitions are operating at frequencies lower than their capacities. After solving the partition LP problems, the maximum of the minimum clock periods computed for the partitions is used to further constrain the top block LP. Consequently, a constraint in the form

$$T \geq \max(T_1, T_2, \ldots T_n) \tag{8.1}$$

is added to the top block LP, where $T_1, T_2, \ldots T_n$ denote the minimum clock periods computed for partitions $LP_1, LP_2, \ldots LP_n$, respectively. If the top block LP problem is less constraining on the minimum clock period compared to the partition LP problems (smaller minimum clock period), then the maximum of the minimum clock periods of the partition LP problems is assigned as the clock period of the top block. Otherwise, the top block LP problem determines the actual minimum operating clock period of the circuit (partitions and top block).

The top block LP problem is solved *after* the partition LP problems are solved, because the top block has the most number of boundary vertices implied in its constraints. Actually, all boundary vertices are implied in the constraints that make up the top block LP problem. Each partition LP problem only has a fraction of the boundary vertices implied in their constraints. The solution of the clock delays to *all* boundary vertices, as computed by each partition LP and the top block LP problems, must match in order to verify the validity of the computed minimum clock period. In order to match these clock delays of boundary vertices, the solutions computed for the top block LP problem are enforced on the partition LP problems with equalities such as:

$$t_i = x_i, \tag{8.2}$$

where the clock delay computed for register R_i in the top block LP problem is x_i time units. If the partition LP problems return feasibility, the computation is complete.

There are two points to note here. First, note that the minimum clock periods computed for partition LP problems are lower limits on the minimum clock period of the complete circuit as each partition LP problem is a subproblem of the original LP problem. The constraints that make up the subproblems are subsets of the LP problem of the complete (original) circuit. As the solution of one of the subproblems (the top block LP problem in this case) is enforced on the remaining LP problems, the convex solution space of the original problem is not violated. Intuitively, therefore, if the presented heuristic method produces a feasible result, this result is optimal.

The second point to note is the fact that the presented heuristic method does not guarantee a feasible solution. The percentage (65%) of ISCAS'89 benchmark circuits for which

the presented heuristic method is feasible are shown in Section 8.3. The following alternative approaches are proposed to solve for cases where the presented heuristic method is not feasible:

- `Reiteration:` The infeasibility diagnostics of an LP solver can be used to resolve the infeasibility problem by changing one or more clock delays that appear in a contradictory constraint. Even if any infeasibility information is not available, iterations can be performed on the infeasible subproblems to search for a feasible answer. The clock delays whose values are changed from the optimal solution of the top block LP are tracked such that the feasibility of the remaining LP problems are not violated. Iterations are performed either until a feasible solution is found or a time limit is reached.
- `Constraining boundary vertices:` As an alternative procedure, the clock delays of all boundary registers can be fixed to a particular value. Synchronous circuits are typically built to operate at zero clock skew, thereby, constraining the clock delays of the boundary vertices to a particular value will guarantee proper circuit operation. The minimum operational clock period for the restricted circuit will be larger than or equal to the minimum clock period of the original circuit due to the additional constraints on the convex solution space. Remember from Section 4.8 that a similar clock delay restriction procedure is applied to the timing of Intellectual Property (IP) blocks. In the experiments performed on ISCAS'89 benchmark circuits for IP blocks, restricting the clock delays of boundary vertices leads to the 27% improvement of conventional clock skew scheduling reduced to 24%, as shown in Table 3 on page 50.
- `Delay padding:` Implementation of clock skew scheduling requires modification of the clock distribution network. If the designers can modify the logic network as well as the clock distribution network, the infeasibility of one or more partition LP problems can be mitigated by delay padding. In this alternative procedure, the data propagation delays of all paths of the infeasible LP problems are formulated as variables. To this end, the minimum D_{Pm}^{if} and maximum D_{PM}^{if} data propagation delays of a local data path can be formulated with additional slack variables S_m^{if} and S_M^{if}, respectively, specific to each local data path. The summation of all the slack variables is added to the minimization type objective function ($\min T$). The coefficient of the minimum clock period T is increased

as appropriate in the LP formulation in order to increase the priority of minimizing T in optimization. In the LP problem solution, the non-zero slack values reported on each local data path are the amounts of delay that must be inserted to the logic paths. The clock delays of the boundary registers must be fixed prior to the solution, such that, the solutions of the remaining LP problems are not violated. The practical concerns of delay padding discussed in Chapter 5 are also valid for this alternative procedure.

8.1.5 Timing-Driven Register Placement

A register placement methodology is presented for the physical design of circuits synchronized with the rotary clocking technology. In this methodology, designated areas for register placement are reserved underneath the ROA rings. Highly populated register banks are stacked inside these designated regions, available for use with the full spectrum of clock phases. Upon synthesis of the circuit and the computation of optimal clock phases, each register in the synthesized netlist is physically mapped to a register underneath the ROA ring. To complete the placement step, the synthesized blocks of combinational circuitry are distributed in the free space inside the region, outside the designated areas.

In Figure 52, an ROA ring of a typical circuit designed with a 0.13 μm technology on a 2mm x 2mm circuit die is illustrated. Note that the figure is not drawn to scale. The die area is evenly divided into 16 regions in a four by four setting, each of which is synchronized with an ROA ring. The dimensions of each ROA ring is 500μm by 500μm. Assuming a single row of registers is placed underneath each ring, the maximum number of registers that are realizable on this die can be easily be obtained using the dimensions of a typical register. In the 0.13um technology, a size of a register is considered to be 4μm by 4μm, with a minimal spacing of 2μm between two instances. Therefore, there is enough space to place approximately 80 registers on each ROA ring edge $[(500+2)/(4+2) \approx 80]$. For 4 sides of an ROA ring and 16 rings, a total of 5120 registers are available for mapping against the synthesized *logic*. This number is adequate for most state-of-the-art digital circuit designs of similar die size. The dimensions of the designated area for register placement and the number of register bank rows are the determining factors for the number of registers in

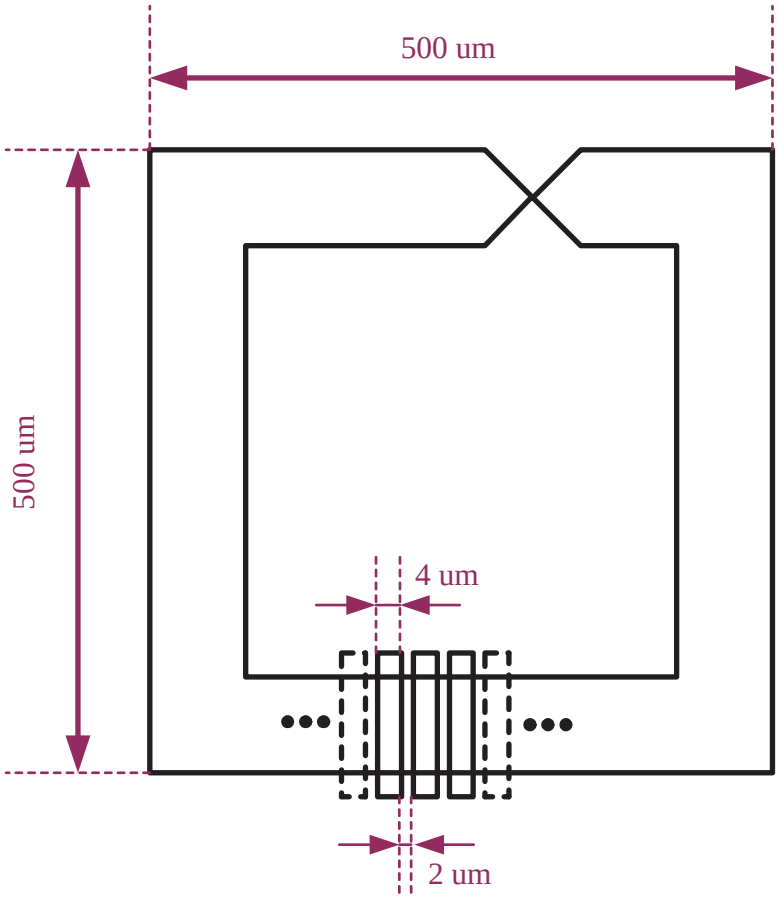


Figure 52: An ROA ring in a chip layout illustrated in 0.13 um technology.

a design, which can be altered for particular design budget requirements. Availability of registers in the register bank enables a good distribution and mapping of clock phases.

The register placement methodology is discussed to demonstrate a viable mechanism to deliver the required clock delays to registers. The implementation of the described register placement methodology is not complete, and they are not an integral part of the advanced timing and synchronization methodologies presented in the dissertation. Thus, the implementation discussion is only included as a direction for future work in this dissertation (Chapter 10).

8.2 COMPUTER-AIDED DESIGN TOOL IMPLEMENTATION

The development of a computer-aided design tool called *hpictiming* following the guidelines of the presented design methodology is in progress in an open source environment [76]. The timing analysis portion of the tool, which is the context of this dissertation, is complete. In this section, the timing portion of the hpictiming tool is presented. The details of the partitioning step implementation with Chaco [31] and clock skew scheduling implementation step with Xgrid parallel computing system are presented. The logic flow of the hpictiming program is presented in Figure 53. This flow is similar to the physical design flow shown in Figure 50 (note that the specific design decisions made in various stages of the implementation of the physical design flow are indicated in detail on Figure 53). In Figure 53, the grid size for partitioning is set to 2x2 for simplicity.

Hpictiming is mainly written in C++ using the standard template library (STL). The code is approximately 250,000 lines. Some of the parsers are written in lex/yacc and the partitioning tool (Chaco software used to implement timing-aware partitioning) is written in ansi C. The program compiles on Gnu/Linux (Debian 3.1), Solaris Unix (Sun OS 9) and Mac OS X 10.3.8 operating systems with the gcc 3.0 (and up) compiler.

The input design data must be in one of two supported formats, either industry standard LEF, DEF and SDF formats or the "bench" format for the ISCAS'89 benchmark circuits. There are two outputs of hpictiming. The first output is the assignment of clock delays to all the synchronous components of an integrated circuit. The second output is the minimum clock period of the circuit, for which the assignment of the clock delays are performed. Several run time input parameters are requested from the user, in order to provide flexibility in run time to perform alternative solution routines (for instance parallel execution of clock skew scheduling on partitions v.s. sequential application on single computing resource). The following are the execution steps of the hpictiming tool:

- Step 1 of 8 – Reading SDF file
- Step 2 of 8 – Reading LEF and DEF files
- Step 3 of 8 – Checking LEF/DEF file consistency
- Step 4 of 8 – Checking SDF file consistency

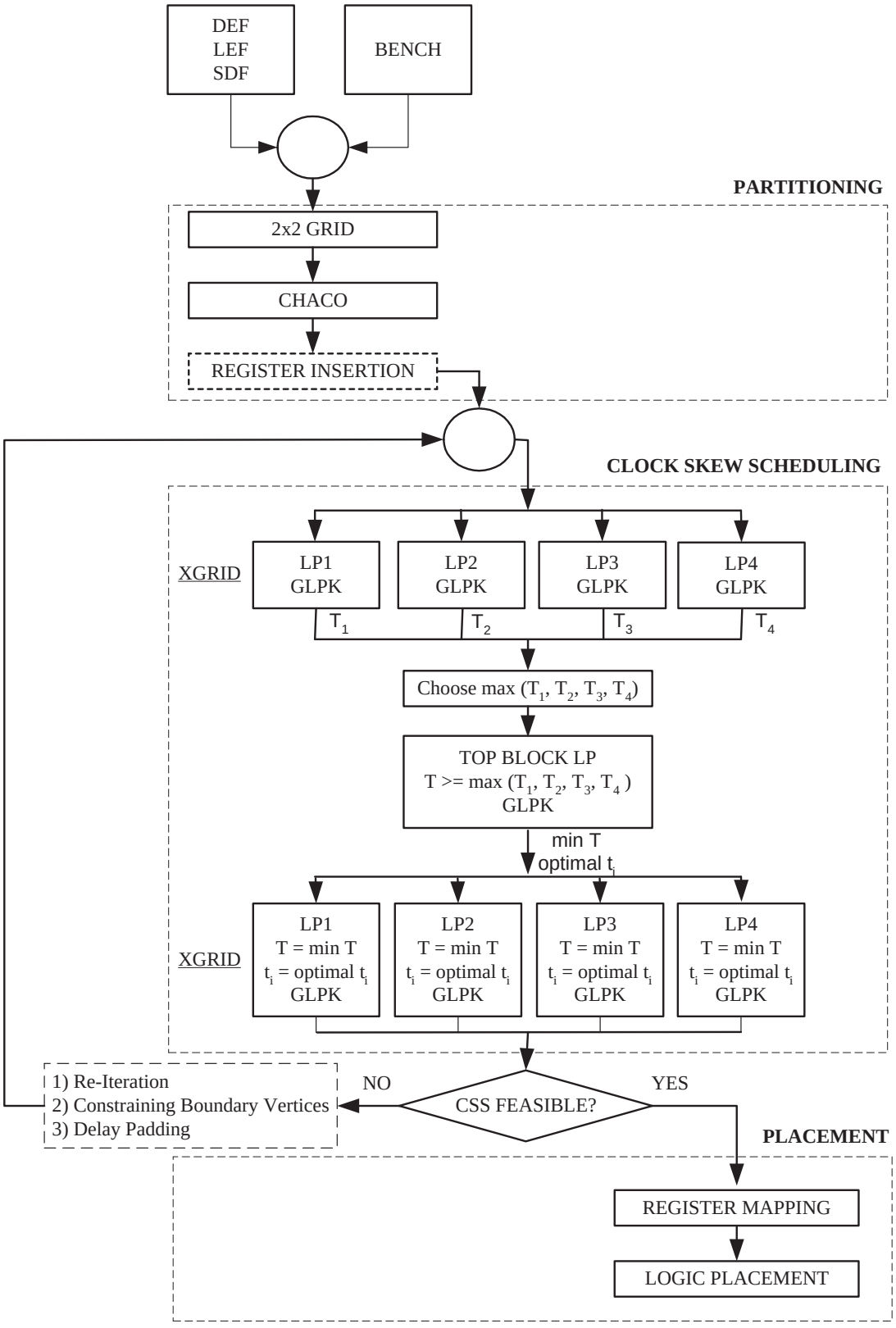


Figure 53: CAD tool flow.

- Step 5 of 8 – Finding directions for each component pin
- Step 6 of 8 – Building connectivity netlist
- Step 7 of 8 – Generating delay information
- Step 8 of 8 – Building register to register paths and output

These 8 steps are manifested for a design input in the industry standard LEF, DEF and SDF formats. After Step 5, the user is prompted to make a selection about partitioning the circuit. If positive answer is received, a fine-tuned partitioning tool Chaco is executed. The primary partitioning method used within the partitioner is a multi-level Kernighan-Lin method [32]. Many parameters provided to the partitioner are not discussed here, but interested readers are referred to Appendix D for a sample execution routine displaying some of these values. After Step 8, the timing of local data paths (register-to-register paths) are obtained, with partitioning information. The user is prompted to make a selection for the solution of the clock skew scheduling problem between a solution on a single computer or a cluster of computers. The application of clock skew scheduling on a single computing resource is straight-forward. Various computing cluster and software alternatives are possible for the parallel implementation of clock skew scheduling. In this work, Apple's Xgrid distributed computing software is selected for implementation.

8.2.1 Parallelization of Clock Skew Scheduling with Xgrid

The popularity of the personal computers in the consumer market over the last few decades has significantly lowered the costs of computing systems. Consequently, the costs associated with setting up a distributed computing system have become relatively affordable. Processes, previously requiring very complex architectures to be executed, can be executed on a cluster of standard computing systems.

Xgrid [5] is a distributed computing software provided by Apple Computers Inc. permitting the operation of a cluster of popular desktop machines as a supercomputer. The Xgrid system aggregates an ad hoc network of Macintosh desktop computers into a *multi-agent computing cluster*, where each agent is called a *computation grid*. Xgrid is typically beneficial for highly parallelized problems that can be broken up into smaller pieces and each

piece executed separately and relatively independent from each other. One of the computers in the cluster is set up as the client for Xgrid and other computers are used as *distributed agents*. The Xgrid software is installed on all computers, enabling the agents to perform grid calculation. The computations can be submitted when the agents are idle or it can be used as the master task. The Xgrid software is run with a *controller*, which regulates the assignment of computing processes to grids and manages the outputs as they are returned to the server. Xgrid software serves as a simple distributed computing infrastructure and does not support message passing between independent agents as is the case for typical Message Passing Interface (MPI) [52] systems.

The parallelization of the application of clock skew scheduling is implemented for the Xgrid distributed computing system. The LP problems for each partition are submitted as individual tasks to the Xgrid computing cluster and solved simultaneously on specific agents. The generated system not only exhibits the pre-described advantages of implementing a parallel execution scheme for clock skew scheduling, it also exemplifies the implementation of a complex VLSI design application on the Xgrid software architecture.

The computing cluster is constructed with eight PowerMac computers with dual G5 1.8GHz microprocessors and 3GB RAM operating Mac OS X 10.3.8. The cluster has one dedicated client, one dedicated controller and six distributed computing agents. The agents are configured to process Xgrid tasks as the master task. This grid computing cluster setup is illustrated in Figure 54.

In order to effectively harness the distributed computing potential, the benchmark circuits are partitioned into four partitions. Note that four partitions emulate a 2x2 grid clock distribution for the rotary clocking technology. The analysis of a 3x3 or a larger grid size is possible, however, perfect parallelization for such grid size can not be achieved with six distributed agents. For the 2x2 grid, four partition LP problems are solved in parallel on four computing agents. The information from the solution of each partition LP is collected on the client computer through writing solutions onto the disk. The following lists the order of execution with collection and distribution of data by the client computer:

- Step 1 – The client executing the *hpictiming* program passes partition LP problems to the controller.

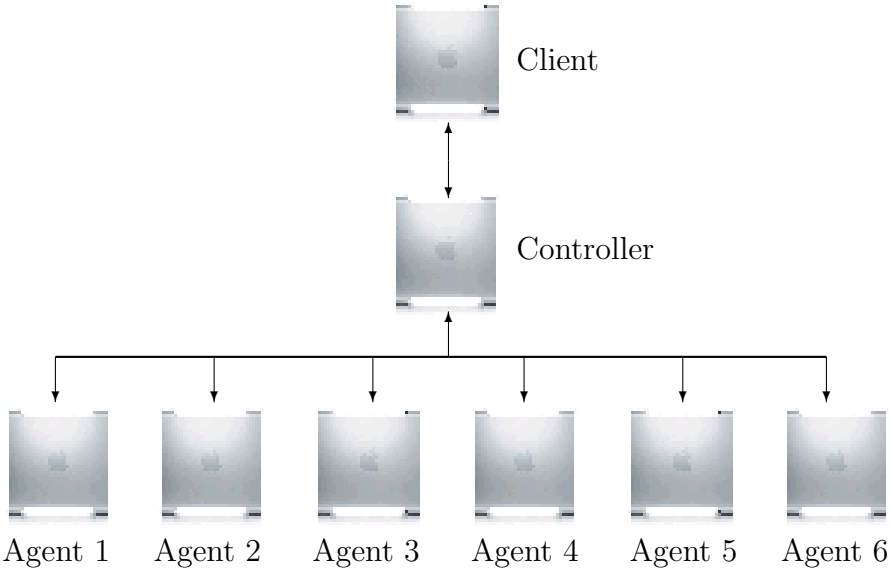


Figure 54: Xgrid computing cluster.

- Step 2 – The controller assigns tasks (solving partition LP problems) to four of the six available agents.
- Step 3 – The agents, which are assigned tasks, complete their tasks. The minimum clock period computed for each task is written to disk and transferred back to the controller.
- Step 4 – Once all agents complete their processes, the data are returned to the client.
- Step 5 – The client processes the data collection, computes the maximum of the minimum clock periods.
- Step 6 – The client appends the constraint to set clock period greater than or equal to the maximum of the minimum clock delays (8.1) to the top partition LP and solves it. The computed clock delays of the form (8.2) and max clock period of (8.1) are appended to the partition LP problems. The client passes the modified partition LP problems to the controller.
- Step 7 – The controller assigns tasks (solving modified partition LP problems) to four of the six available agents.

- Step 8 – The agents, which are assigned tasks, complete their tasks. The solutions are written to disk and returned to the controller.
- Step 9 – Once all agents complete their processes, the data are returned to the client.
- Step 10 – The client, executing *hpictiming*, processes the data collection to identify feasibility.

8.2.2 Speedup of Computation

The primary advantage gained from the parallelization of the application of clock skew scheduling is the speedup in computation time. The speedup is gained not only from parallelization but also from partitioning the LP problems. In this section, Amdahl's law [3] is used to compute the speedups achieved through partitioning and parallelization in the application of clock skew scheduling. Amdahl's law states that, the performance gain that can be obtained by improving some fraction of a task depends on the following formula:

$$Speedup = \frac{1}{(1-f) + \frac{f}{s}} \tag{8.3}$$

where, f is the fraction of a task that is enhanced, and s is the speedup of the enhanced portion.

In the physical design flow shown in Figure 50, the clock skew scheduling step can be parallelized. However, parallelization requires at least two stages of execution due to the required iterations between the top block and partition LP problems (Figure 53). The two stages of execution inhibits the derivation of a simple mathematical expression for the speedup of the enhanced portion s. Thus, instead of identifying the portions which can be parallelized and computing the speedups offered in each iteration, the following simpler form of Amdahl's law is used:

$$Speedup = \frac{\text{Run time for entire task without using enhancement}}{\text{Run time for entire task using the enhancement when possible}}. \tag{8.4}$$

The formula given in (8.3) is used in the evaluation of results in Section 8.3.

The simple and intuitive formula of (8.4) is used to compute the speedups achieved through partitioning and parallelization. The formula adapted to the parallel application of clock skew scheduling to partitions is:

$$Speedup = \frac{\text{Run time of physical design flow without partitioning}}{\text{Run time of physical design flow with partitioning and parallelization}}. \quad (8.5)$$

In a distributed computing environment, communication overhead is often of concern. In the Xgrid environment, because of the simple (practically non-existent) interface between independent computing agents, the communication overhead is reduced to a minimum. The bulk of the required communication occurs in distributing the tasks to agents. The communication times are included in the run times. However, the communication times are not explicitly reported because of their insignificance compared to total computation times.

8.3 EXPERIMENTAL RESULTS

In this section, the experimental results for the operation of rotary clocking technology and the application of *hpictiming* CAD tool are presented. The technical details in the implementation and operation of rotary clocking technology [96, 97] are outlined here for completeness of the discussion of rotary clocking technology. The main focus in experimentation in this section is on the effectiveness of the application of clock skew scheduling with partitioning (sequentially and in parallel). Towards this goal, the quality of the partitioning process, the run times for the parallelized application of clock skew scheduling on the parallel computing clusters and final circuit performances are analyzed.

8.3.1 Rotary Clocking Results

The operation of the ROA structure in providing a gigahertz frequency, low jitter, low power clock signal with fast transition times is confirmed by simulating the ring shown in Figure 46 at 965MHz and 3.4GHz. The ring designed at 2.5V 0.25 μm CMOS technology has 25 interconnected RTWO rings on a 7x7 array grid. The simulation result presented in [96] for

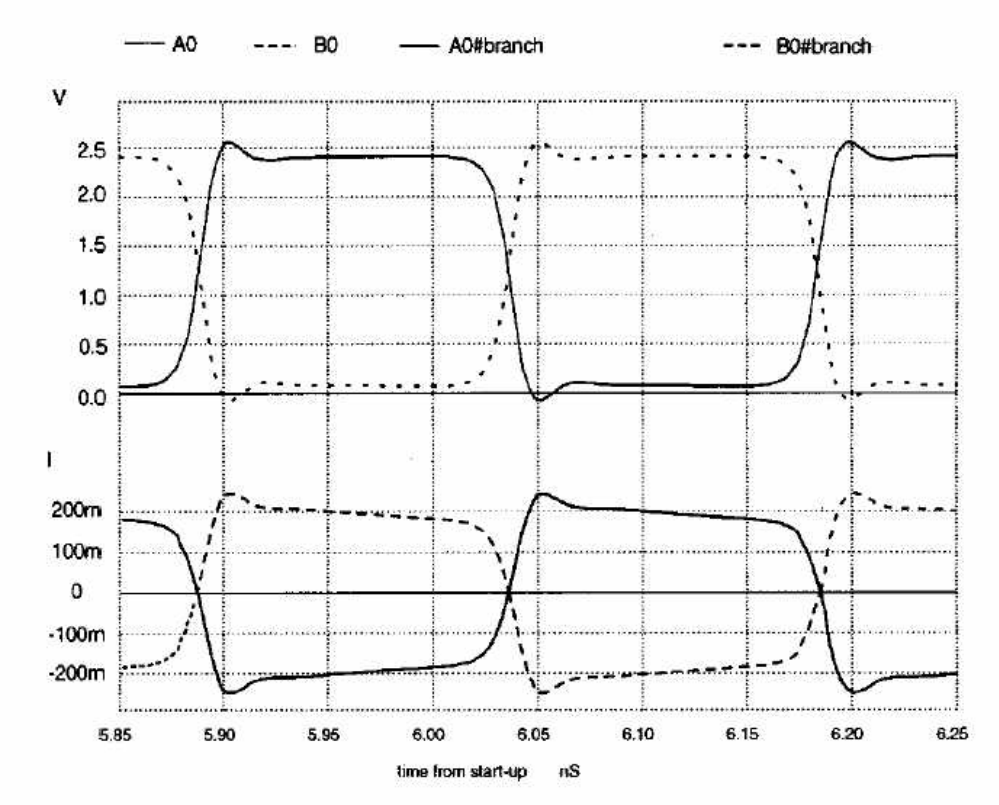


Figure 55: Line voltage and line current for the 3.4GHz clock example.

the 3.4GHz ring is shown in Figure 55. Very promising results of a clock jitter of 5.5ps and 34-dB power supply rejection ratio (PSRR) are measured [96].

Two other important metrics for an oscillator are the sensitivity to changes in temperature and supply voltage. It has been shown that the frequency deviation with temperature change between -50^oC and 150^oC is only 1% while the change with V_{DD} deviation between 1.5 and 3.5V is around 2% [96]. The immunity of the RTWO signals to process variations while allowing full skew control over 360 degree phases on the ring proves very valuable for deep sub micrometer applications.

8.3.2 Circuit Partitioning Results

The Chaco partitioning tool is tested on real and artificially generated circuits with various netlist sizes. A moderate size circuit of over 80000 gates and 4000 registers is artificially created in order to test the effectiveness of Chaco. Partitioning of this circuit into a 10x10 grid of 100 ROA rings is completed under 5 seconds on a 2.4GHz AMD processor workstation with 1GB RAM. The resulting partitions have 60% of nodes as internal vertices and an even distribution of approximately 800 nodes per partition.

More detailed results are reported for the ISCAS'89 benchmark circuits and for one industrial circuit after processing these circuits with the *hpictiming* tool. The largest of these real circuits is the industrial circuit `industrial1` with 107875 circuit components, including 14031 synchronous registers. Path enumeration of `industrial1` cannot be completed within the available computing resources. Thus, gate-level partitioning is performed as discussed in Section 8.1.2. The ISCAS'89 benchmark circuits are sufficiently large, yet, path enumeration can be successfully performed on these circuits. Thus, partitioning is applied to the register-transfer level netlist for these circuits. The partitioning results for

- ISCAS'89 benchmark circuits, with register-transfer level netlist partitioning,
- `Industrial1` circuit, with gate level netlist partitioning,

are reported as computed on an PowerMac computer with dual G5 1.8GHz microprocessors and 3GB RAM running Mac OS X 10.3.8.

As discussed in Section 8.1.2, register insertion is necessary for gate level netlist partitioning. In order to profile the number of the inserted registers for different partitioning grid sizes, experiments are performed on `industrial1`. The circuit statistics for `industrial1` are as follows:

```
Number of Components                    :   107875
Number of Synchronous Components        :    14031
```

Note that the number of paths in this circuit are not known because of the failure of path enumeration. It is known, however, that the number of paths exceed 12 million, before the process execution of path enumeration terminates.

Table 17: Number of inserted registers for `Industrial1` after partitioning.

Grid (Partition) Size	2x2	4x4	5x5	6x6	10x10
Number of Inserted Registers	3011	9751	13903	16172	32131

The circuit statistics after partitioning `industrial1` into grid sizes of 2x2, 4x4, 5x5, 6x6 and 10x10 are presented in Table 17. In Table 17, the entry "Number of Inserted Registers" is the number of level-sensitive latches operating in the transparent phase of operation that are inserted on cut edges. It is observed that as the number of partitions increases, the number of registers that need to be inserted on edges (for non-register input cuts) increases. For a 5x5 sized partition, the number of inserted registers (13903) approaches the number of registers already present in the circuit (14031). This increase in the number of inserted registers makes implementation with higher number of partitions impractical in most cases.

As described in Section 8.1.2, the selection of parameters for Chaco favors the interval vertices to be high and boundary vertices and edge cuts to be low. For proper synchronization, the set sizes must not be extremely unbalanced. Listed in Figure 56 are results of the Chaco partitioner for `industrial1`. In Figure 56, the entry "Set Size" refers to the number of vertices in a partition. The set sizes are less than the number of total components because the test circuitry are not considered in the partitioning process. The entry "Edge Cuts" is the number of cuts over data paths. Edge cuts include the encouraged cuts on register inputs and the cuts between two logic gates, which require register insertion. The entries "Boundary Vertices" and "Internal Vertices" *refer* to components that communicate and do not communicate with other partitions, respectively (Section 8.1.3). The minimum and maximum value columns report the minimum and maximum value over all partitions for the corresponding row.

Note that the number of vertices is not the number of unique boundary vertices as the name may imply. The number of vertices reflects the total weight of vertices. In Chaco, each vertex is weighed identically with unity weight. During partitioning, the weight of a

```
                        Total      Max/Set      Min/Set
                        -----      -------      -------
Set Size:               65908        16486        16470
Edge Cuts:             255516       136769       113643
Boundary Vertices:       7900         2178         1786          Partition Size: 2x2
Internal Vertices:      59393        14859        14838

Run Time:               15.29 secs


                        Total      Max/Set      Min/Set
                        -----      -------      -------
Set Size:               65908         4314         4049
Edge Cuts:             674371       193668        10924
Boundary Vertices:      21541         1719          555          Partition Size: 4x4
Internal Vertices:      49385         3647         2896

Run Time:               22.85 secs


                        Total      Max/Set      Min/Set
                        -----      -------      -------
Set Size:               65908         2744         2554
Edge Cuts:             999657       136127         7603
Boundary Vertices:      27752         1478          418          Partition Size: 5x5
Internal Vertices:      45616         2252         1659

Run Time:               27.78 secs


                        Total      Max/Set      Min/Set
                        -----      -------      -------
Set Size:               65908         1898         1817
Edge Cuts:         1.21005e+06       177341         6303
Boundary Vertices:      34141         1474          138          Partition Size: 6x6
Internal Vertices:      42857         1710         1051

Run Time:               32.14 secs


                        Total      Max/Set      Min/Set
                        -----      -------      -------
Set Size:               65908          693          583
Edge Cuts:         2.28313e+06       146214         2808
Boundary Vertices:      63686         1110          143        Partition Size: 10x10
Internal Vertices:      27787          583          170

Run Time:               54.08 secs
```

Figure 56: Chaco outputs for circuit `Industrial1`.

vertex is re-evaluated depending on the number of edge cuts. For instance, if a boundary vertex has two inter-partition paths, the weight of that boundary vertex is set to two. Such definition of the number of "Boundary Vertices" is suitable for timing-driven partitioning. However, the users must be aware that the reported numbers are not the total number of boundary vertices of a partition. The actual number of boundary numbers is smaller than the "Boundary Vertices" reported by Chaco. Note that the number of boundary vertices must match "Set Size - Internal Vertices".

Chaco partitioner is applied to ISCAS'89 benchmark circuits, at the register-transfer level. Partitioning on this level is preferred (where applicable) because register insertion is not necessary. The run times and partitioning results are presented in Table 18. The run times are reported only for the execution of Chaco, and are almost negligible when compared to actual path enumeration or hpictiming total run times. Overall, Chaco generates partitions that are well suited for clock skew scheduling and require reasonable run times even for large-scale circuits.

8.3.3 Clock Skew Scheduling of Partitions Results

Clock skew scheduling is applied in parallel to the partitions of ISCAS'89 benchmark circuits and the industrial circuit `industrial1`. The partitioning results, presented in Section 8.3.2, are utilized within hpictiming in generating the top block and the partition LP problems. The LP problem [21] shown in Table 6 (page 69) is used for clock skew scheduling of edge-sensitive synchronous circuits. For `industrial1`, where register insertion is performed, linear constraints described in Chapter 4 for level-sensitive local data paths are added to the LP problem constraints. The feasibility of the parallel application of clock skew scheduling is analyzed. The speedups achievable through parallel clock skew scheduling are computed.

The experimental setup of Chapters 4 and 5 is replicated for experimentation. The timing information for each circuit component is generated with a pre-determined algorithm, accounting for fanouts of a component, and the size and the type of the component. A 50% duty cycle single phase clock signal is selected. The internal register delays are assumed to be zero $\left(S_f = H_f = D^i_{CQ} = D^i_{DQ} = 0\right)$. The experiments are performed on an Xgrid cluster

Table 18: Chaco 2x2 partitioning results on ISCAS'89 benchmark circuits.

Circuit	Set Size	Edge Cuts	Boundary Vertices	Internal Vertices	Run Time (secs)
s27	3	3	6	0	0.03
s208.1	8	24	24	0	0.02
s298	14	43	35	1	0.04
s344	15	50	36	0	0.04
s349	15	50	36	0	0.04
s382	21	78	46	0	0.04
s386	6	13	18	0	0.02
s400	21	78	46	0	0.03
s420.1	16	96	48	0	0.02
s444	21	74	51	1	0.02
s510	6	13	18	0	0.03
s526	21	81	47	0	0.03
s526n	21	81	47	0	0.04
s641	19	56	38	2	0.03
s713	19	56	38	2	0.04
s820	5	9	15	0	0.03
s832	5	9	15	0	0.02
s838.1	32	384	96	0	0.05
s938	32	384	96	0	0.06
s953	29	114	37	4	0.04
s967	29	114	37	4	0.03
s991	19	42	23	2	0.04
s1196	18	11	12	7	0.03
s1238	18	11	12	7	0.01
s1423	74	1003	185	7	0.10
s1488	6	13	18	0	0.02
s1494	6	13	18	0	0.01
s1512	57	196	124	1	0.05
s3271	116	358	141	22	0.08
s3330	132	156	96	48	0.08
s3384	183	660	174	50	0.13
s4863	104	380	161	17	0.06
s5378	179	542	271	36	0.12
s6669	239	455	237	56	0.13
s9234	228	824	319	34	0.18
s9234.1	211	684	248	55	0.16
s13207	669	652	447	348	0.13
s15850	597	7025	858	153	0.41
s15850.1	534	5787	773	140	0.64
s35932	1728	167	258	1506	0.41
s38417	1636	1692	688	997	1.28
s38584	1452	2935	1414	524	1.07

built with eight PowerMac computers with dual G5 1.8GHz microprocessors and 3GB RAM running Mac OS X 10.3.8 (Section 8.2.1). The simplex optimizer of the GNU LP solver GLPK (version 4.8) [25] is used to solve the LP problems. The results are presented on Table 19. In Table 19, the number of registers r and the number of paths p are shown for each analyzed circuit. Run times of various clock skew scheduling methods are shown. Run times of the conventional method of Table 6 are denoted by t_{conven}, the run times of the sequential solution of partitions method are denoted by t_{sequen} and the run times of the parallel solution of partitions method are denoted by t_{paral}. The feasibility of each circuit when solved with the presented heuristic method is shown on the column labeled "Feasibility".

The minimum clock periods computed via each of the three methods (when feasible) are identical, and equal to the values reported in Tables 3 and 9 under columns T_{FF}^{CSS}. These minimum clock periods, as presented in [43] (and repeated in Tables 3 and 9) provide an average of 30% improvement over conventional zero clock skew, edge-triggered circuits. In this section, the target is to improve the run times of clock skew scheduling without deteriorating these clock period improvements. Accordingly, the run times in Table 19 are reported in order to demonstrate the speedups achievable through partitioning and parallel application of clock skew scheduling.

The selected suite of ISCAS'89 benchmark circuits and `industrial1` are partitioned into a 2x2 partition using Chaco. The partition and top block LP problems are generated. First, the generated LP problems are solved on a single workstation in a sequential order. The observed run times t_{sequen} record speedups over conventional clock skew scheduling application due to partitioning. Second, the generated LP problems are solved on the Xgrid computing cluster in parallel as described in Section 8.2.1. The observed run times t_{paral} record speedups over the conventional clock skew scheduling application due to partitioning and parallelization of the application. Note that the application of clock skew scheduling to `industrial1` using the conventional clock skew scheduling method is not possible, thus run times are not reported.

It is observed from Table 19 that t_{paral} is consistently and significantly (especially for large scale circuits) superior to t_{sequen} and t_{conven}. Similarly, t_{sequen} is consistently superior

Table 19: Clock skew scheduling results on 2x2 partitioned ISCAS'89 circuits.

Circuit Info			**Run Time CSS (sec)**			**RTI (%)**		**Feasibility**
Circuit	r	p	t_{conven}	t_{sequen}	t_{paral}	RTI_{sequen}	RTI_{paral}	Feasibility
s27	3	4	0	0	0	0	0	yes
s208.1	8	28	0	0	0	0	0	yes
s298	14	54	0	0	0	0	0	yes
s344	15	68	0	0	0	0	0	yes
s349	15	68	0	0	0	0	0	yes
s382	21	113	0	0	0	0	0	yes
s386	6	15	0	0	0	0	0	yes
s400	21	113	0	0	0	0	0	yes
s420.1	16	120	0	0	0	0	0	no
s444	16	113	0	0	0	0	0	yes
s510	6	15	0	0	0	0	0	yes
s526	21	117	0	0	0	0	0	yes
s526n	21	117	0	0	0	0	0	yes
s641	19	81	0	0	0	0	0	no
s713	19	81	0	0	0	0	0	no
s820	5	10	1	1	1	0	0	yes
s832	5	10	0	0	0	0	0	yes
s838.1	32	496	2	0	0	0	100	no
s938	32	496	1	1	1	0	0	no
s953	29	135	0	0	0	0	0	yes
s967	29	135	0	0	0	0	0	yes
s991	19	51	0	0	0	0	0	yes
s1196	18	20	0	0	0	0	0	no
s1238	18	20	0	0	0	0	0	no
s1423	74	1471	21	6	3	71	86	yes
s1488	6	15	0	0	0	0	0	yes
s1494	6	15	0	0	0	0	0	yes
s1512	57	415	1	0	0	100	100	yes
s3271	116	789	4	2	1	50	75	no
s3330	132	514	2	2	1	0	50	no
s3384	183	1759	22	4	3	82	86	yes
s4863	104	620	2	0	0	100	100	yes
s5378	179	1147	9	5	2	44	78	no
s6669	239	2138	33	10	7	30	79	no
s9234	228	247	52	15	8	71	85	no
s9234.1	211	2342	47	12	5	74	89	yes
s13207	669	3068	86	17	10	80	88	yes
s15850	597	14257	3545	735	447	79	87	no
s15850.1	534	10830	1358	156	110	89	92	yes
s35932	1728	4187	101	38	13	62	87	no
s38417	1636	28082	7707	3780	1845	51	76	yes
s38584	1452	15545	1394	749	339	46	76	yes
Industrial1	14031	3692878	n/a	34680	25680	n/a	n/a	no
Average						25	28	

to t_{conven}. The run time improvement from t_{conven} to t_{sequen} and from t_{conven} to t_{paral} are listed under RTI_{sequen} and RTI_{paral}, respectively. The improvements are computed with the formula $[100(t_{old} - t_{new})/t_{old}]$. On the ISCAS'89 benchmark circuits, the average run time improvement via partitioning (RTI_{sequen}) is 25%. The average run time improvement via partitioning and parallel application of clock skew scheduling RTI_{paral} is 28%. The circuits, for which the method is infeasible, are not considered in the computations of the average improvement. Overall, the application of clock skew scheduling to partitions is feasible for 28 (65%) of the total 43 circuits, whereas this method is not applicable to the remaining 15 circuits (35%). For these 15 circuits, the alternative methods described in Section 8.1.4 can be used.

8.3.4 Overall CAD Tool Results

In this section, the run times of hpictiming on the benchmark circuits are analyzed to profile the speedups gained in overall program execution due to partitioning and parallelization. In particular, the speedups available through solving the partition problems sequentially and in parallel are computed using Amdahl's Law presented in (8.5).

Table 20 presents the speedup results of *hpictiming* tool on ISCAS'89 benchmark and `Industrial1` circuits. In Table 20, the number of registers and paths of each circuit are shown with r and p, respectively. Run times of the hpictiming tool operated with various clock skew scheduling methods on the ISCAS'89 benchmark circuits are shown. Run times of hpictiming with the conventional clock skew scheduling method of Table 6 are denoted by $t_{conven}^{hpictiming}$, the run times with the sequential solution of partitions method are denoted by $t_{sequen}^{hpictiming}$ and the run times with the parallel solution of partitions method are denoted by $t_{paral}^{hpictiming}$. In Table 20, the speedups due to partitioning and sequential application of clock skew scheduling to 2x2 partitions of the circuits are denoted by $speedup_{sequen}$. The speedup $speedup_{sequen}$ is computed with the following formula:

$$speedup_{sequen} = \frac{t_{conven}^{hpictiming}}{t_{sequen}^{hpictiming}}. \tag{8.6}$$

Table 20: Speedup of hpictiming on 2x2 partitioned ISCAS'89 circuits.

Circuit Info			**Run Time hpictiming (sec)**			**Speedup (X)**	
Circuit	r	p	$t_{conven}^{hpictiming}$	$t_{sequen}^{hpictiming}$	$t_{paral}^{hpictiming}$	$speedup_{sequen}$	$speedup_{paral}$
s27	3	4	0	0	0	n/a	n/a
s208.1	8	28	0	0	0	n/a	n/a
s298	14	54	0	0	0	n/a	n/a
s344	15	68	0	0	0	n/a	n/a
s349	15	68	1	1	1	1.0x	1.0x
s382	21	113	0	0	0	n/a	n/a
s386	6	15	0	0	0	n/a	n/a
s400	21	113	0	0	0	n/a	n/a
s420.1	16	120	0	0	0	n/a	n/a
s444	16	113	1	1	1	1.0x	1.0x
s510	6	15	1	1	1	1.0x	1.0x
s526	21	117	0	0	0	n/a	n/a
s526n	21	117	1	1	1	1.0x	1.0x
s641	19	81	1	1	1	1.0x	1.0x
s713	19	81	0	0	0	n/a	n/a
s820	5	10	1	1	1	1.0x	1.0x
s832	5	10	0	0	0	n/a	n/a
s838.1	32	496	3	1	1	3.0x	3.0x
s938	32	496	1	1	1	1.0x	1.0x
s953	29	135	1	1	1	1.0x	1.0x
s967	29	135	1	1	1	1.0x	1.0x
s991	19	51	1	1	1	1.0x	1.0x
s1196	18	20	0	0	0	n/a	n/a
s1238	18	20	1	1	1	1.0x	1.0x
s1423	74	1471	22	7	4	3.1x	5.5x
s1488	6	15	1	1	1	1.0x	1.0x
s1494	6	15	1	1	1	1.0x	1.0x
s1512	57	415	2	1	1	2.0x	2.0x
s3271	116	789	6	4	3	2.5x	2.0x
s3330	132	514	2	2	1	1.0x	2.0x
s3384	183	1759	25	7	6	3.6x	4.2x
s4863	104	620	6	4	4	2.5x	2.5x
s5378	179	1147	15	11	8	1.4x	1.9x
s6669	239	2138	40	17	14	2.4x	2.9x
s9234	228	247	60	23	16	2.6x	3.8x
s9234.1	211	2342	53	18	11	2.9x	4.8x
s13207	669	3068	105	36	29	2.9x	3.6x
s15850	597	14257	3757	947	659	4.0x	5.7x
s15850.1	534	10830	1385	185	138	7.5x	10.0x
s35932	1728	4187	313	250	225	1.3x	1.4x
s38417	1636	28082	7881	3958	2021	2.0x	3.9x
s38584	1452	15545	1615	1022	611	1.6x	2.6x
Industrial1	14031	3692878	n/a	36062	27046	n/a	n/a
Average						2.1x	2.6x

The speedups due to partitioning and application of clock skew scheduling in parallel are denoted by $speedup_{paral}$. The speedup $speedup_{parallel}$ is computed with the following formula:

$$speedup_{paral} = \frac{t_{conven}^{hpictiming}}{t_{paral}^{hpictiming}}. \tag{8.7}$$

Remember from Section 8.3.3 that the application of clock skew scheduling with partitioning is not feasible for some of the ISCAS'89 benchmark circuits and the industrial circuit `industrial1`. The circuits for which the method is not applicable are not considered in the computation of average speedups. Still, the speedup numbers are presented individually for all the ISCAS'89 benchmark circuits and the industrial circuit `industrial1` in Table 20.

It is observed from Table 20 that on average 2.1x speedup is observed in hpictiming run time due to partitioning. If the partitioned LP problems are solved in parallel, the average speedup is 2.6x. It is intuitive that as the size of a circuit increases, the clock skew scheduling step of hpictiming, which is the fraction of the task that is enhanced with partitioning and parallelization, increases as well. So, for larger size circuits, higher values of speedup are expected through partitioning and parallelization. Indeed, such a trend is observed in Table 20.

Amdahl's law of (8.3) is further investigated on several of the benchmark circuits. The execution of hpictiming is divided into three main steps, *Read-in*, *Partitioning* and *Scheduling*. The *Read-in* step consists of reading the input data and identifying the local data paths. Steps 1 through 8 presented in Section 8.2 constitute the read-in step. *Partitioning* step consists of the timing-driven partitioning procedure implemented with Chaco, discussed in Section 8.1.2. *Scheduling* step consists of the application of clock skew scheduling to generated partitions.

Figure 57 illustrates the relative run time lengths of each step for several ISCAS'89 benchmark circuits and the industrial circuit `industrial1` for the parallel application of clock skew scheduling. The ISCAS'89 benchmark circuits, whose total run times are below a certain limit, are not included in the analysis. The selectivity about the ISCAS'89 benchmark circuits is to eliminate the inaccuracies due to the rounding off errors in run times, most prominent for circuits with a run time below a few seconds. Although the solution for `industrial1` is infeasible, the reported run times are believed to be a good approximation

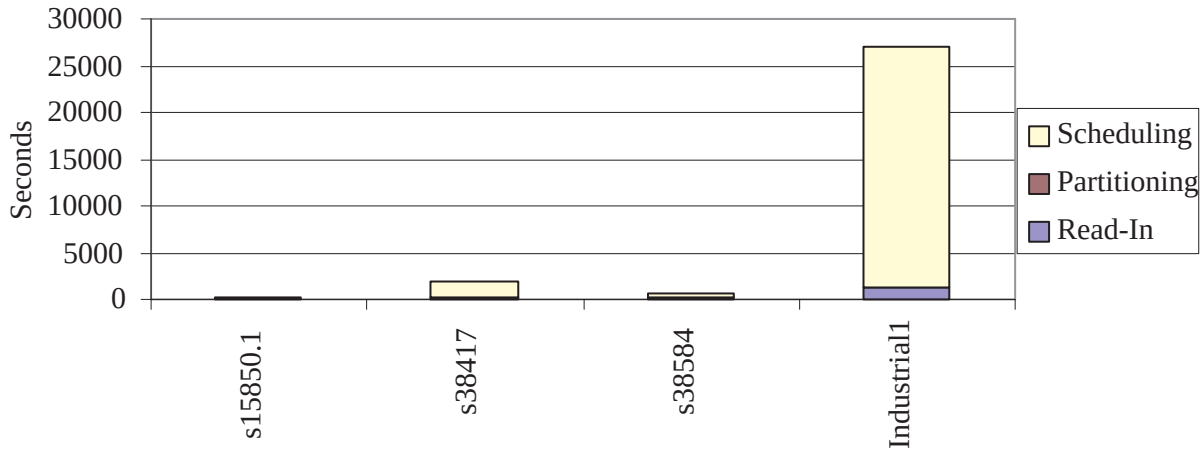


Figure 57: The run times of hpictiming with Xgrid on large circuits.

of what they would have been, if all the subpartitions had been feasible. Note that run times illustrated in Figure 57 for *partitioning* and *scheduling* steps are presented in Tables 18 and 19. In particular, *partitioning* step run time is presented in Tables 18 under column "Run Time" and *scheduling* step run time is presented in Table 19 under column t_{paral}. The total run time of the hpictiming program (with parallel application of clock skew scheduling) is reported in Table 20 under the column $t_{paral}^{hpictiming}$.

The breakdown of run times to the three steps of hpictiming is shown for the three largest circuits, `s38584`, `s38417` and `Industrial1`. The run times are shown in Figure 58, 59 and 60 for `s38584`, `s38417` and `Industrial1`, respectively.

The run times for three application methods—conventional, sequential and parallel application of clock skew scheduling—are shown for each circuit. The run times for each step of hpictiming is shown with color codes, listed as read-in, partitioning and scheduling steps from bottom to top for each data bar.

Partitioning step is not required in the conventional application method, thus is not shown on the run time bar in the figures for the conventional application cases. Even for methods where partitioning is necessary, the partitioning stage of the run time bar is

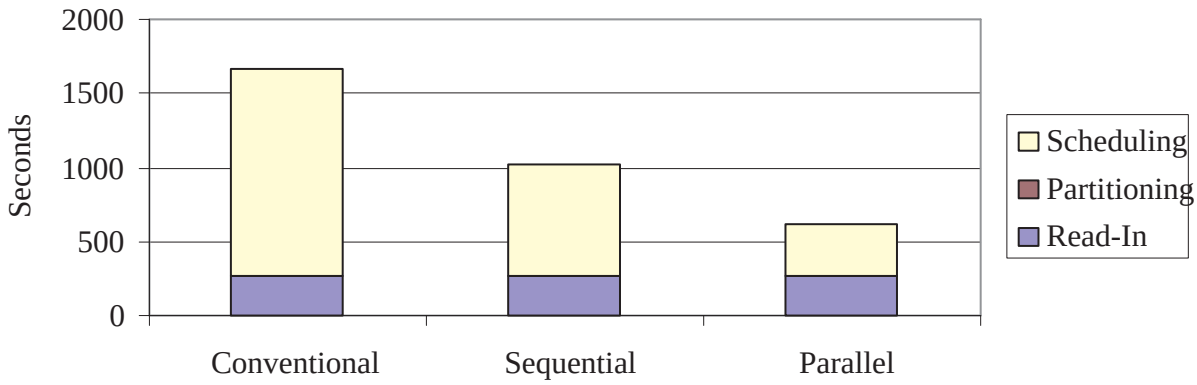


Figure 58: Run time breakdown of hpictiming program steps for s38584.

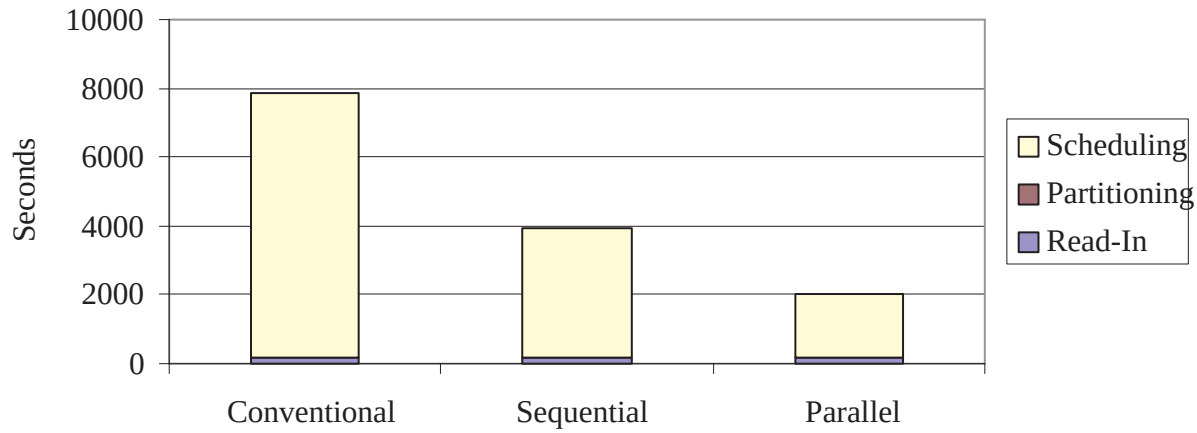


Figure 59: Run time breakdown of hpictiming program steps for s38417.

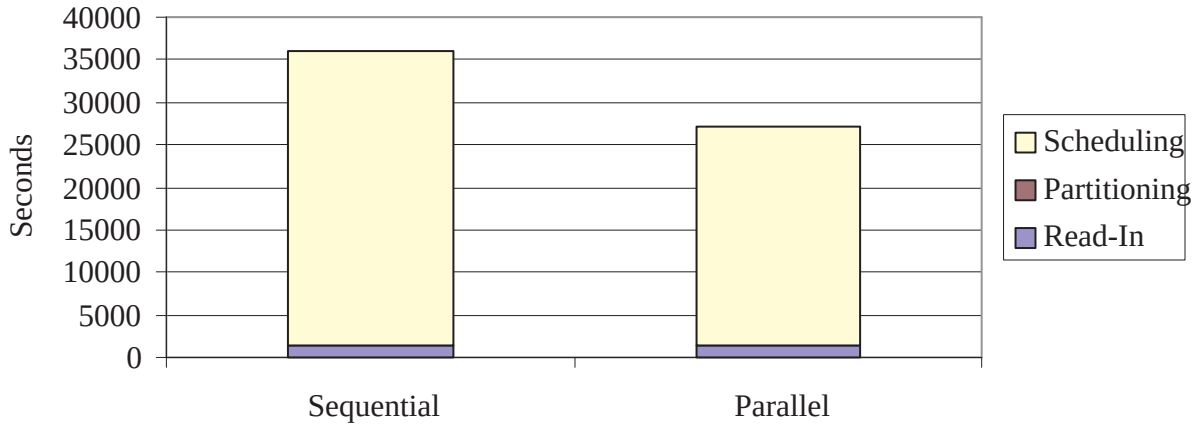


Figure 60: Run time breakdown of hpictiming program steps for industrial1.

not visible, because the run times for the partitioning process with Chaco are very small compared to the rest of the execution time.

Note that the run time of the read-in and partitioning (where applied) steps are identical in all three application methods. Through partitioning and application of clock skew scheduling in parallel, the run time of the clock skew scheduling step of the hpictiming program is improved. This improvement speeds up the hpictiming program according to Amdahl's law, the results of which are presented in Table 20.

8.4 SUMMARY

In this chapter, a design methodology for the physical design of digital VLSI circuits synchronized with the resonant rotary clocking technology is described. The presented physical design flow is similar to conventional physical design flows, in that, it has partitioning, timing and placement steps. The implementation of a CAD tool based on the presented physical design flow is described. The CAD tool is demonstrated to be functional and efficient, lead-

ing to improvements in the performance of the designed circuits and the software run time. The scalability of the application of clock skew scheduling is particularly improved through partitioning and parallelization. The finalized CAD tool can be used as a blueprint of future CAD tools used for industrial-strength applications.

The preliminary results of this work are published in [86]. The hpictiming CAD tool is available under general public license (GPL) [76].

9.0 CONCLUSIONS

The design challenges of nano-scale CMOS and emerging nano-electronics structures lead to a different set of paradigms for circuit designers. The challenges stemming from the necessity to design, analyze and verify complex giga-scale systems with increasingly significant manufacturing variations create multi-disciplinary research areas. Systems are designed with more awareness towards the physical laws and stochastic modeling, while the design budgets are becoming critically narrow. Design techniques are shifting to yield intelligent fault detection and design for manufacturing techniques, while the application areas are constantly spreading over to new disciplines.

The increasingly important consequences of the dominance of the physical phenomena in nano-scale CMOS design have affected the integrated circuit design flow. Engineers are required to carefully investigate the physical behavior of materials and use sophisticated techniques to design products in these ultra large scales. Some previously ignored material behavior leads to a design altering phenomena at these larger scales. However, these material behavior can also be used to derive novel design techniques. The oscillatory properties of signals over long wires of interconnect, for instance, are used in designing the resonant clocking technologies.

Within such a dynamic and resourceful environment, where design techniques are evolving to adapt to nano-scale silicon implementations, Computer-Aided Design (CAD) tools are crucial to the successful development and integration of novel design techniques. The adaptation to the shifts in the design paradigms will be realizable only through advances in CAD. In this dissertation, novel approaches in the CAD development for the advanced timing and synchronization of complex giga-scale systems are presented. The proposed design automation approaches are used as an integral part of the physical design flow for a set of

next-generation integrated circuits, which are synchronized by the rotary clocking technology. The clock skew scheduling and rotary clocking technologies, which do not seamlessly blend with the traditional design flow of integrated circuits, are shown to constitute a very efficient circuit implementation alternative for next-generation circuits. Experimental results project significantly superior circuit implementations for non-zero clock skew, rotary clocking synchronized circuits—demonstrating lower power consumption, higher operating speed and increased tolerance to process parameter variations. The integration of the proposed CAD algorithms and the clocking technology for nano-scale CMOS circuits are instrumental to demonstrate how the CAD tools potentially enable and propel the technological development of integrated circuit design.

In Chapter 4, a linear programming formulation for the static timing analysis of level-sensitive circuits is described. This LP formulation is the first stand-alone formulation offered for the timing analysis of non-zero clock skew, level-sensitive circuits. The majority of the current static timing analyzers utilize iteration-based approaches to analyze the timing behavior of systems with latches. These iteration-based approaches are shown to converge to solutions relatively quickly for most circuits, however, they require algorithmic extensions for complex circuit topologies. The LP formulation presented in this dissertation is topologically independent and shown to operate with reasonable run times.

The LP formulation presented in Chapter 4 is also used to automate the application of clock skew scheduling to level-sensitive circuits for the first time. The clock skew scheduling of edge-sensitive circuits has previously been exhaustively investigated by researchers. However, clock skew scheduling of level-sensitive circuits has not been successfully addressed. The relatively higher complexity of the analysis of level-sensitive circuits and the lack of an elegant automation framework (iterative solution procedures are less flexible) for such circuits have inhibited the automation of clock skew scheduling. The clock skew scheduling problem of level-sensitive circuits is formulated on the presented LP framework. Performance improvements of 27% shorter clock periods on average are obtained for non-zero clock skew, level-sensitive circuits over traditionally used zero clock skew, edge-sensitive circuits. The solutions of the LP problems are empirically shown to be optimal with experiments on the ISCAS'89 suite of benchmark circuits.

In Chapter 5, the optimal clock schedules and data propagation times of a circuit are analyzed after clock skew scheduling. With these analyses, the theoretical limits of improvement in the minimum clock period achievable through clock skew scheduling are identified. Prior to this study, it has been considered that the data path cycles and delay uncertainties are the only limiting factors on the minimum clock period achievable through clock skew scheduling. Many static timing analysis tools and algorithms proposed in the last two decades have been operating on this principle. In this research, it is shown that the reconvergent data paths also introduce theoretical limits on the minimum achievable clock period through clock skew scheduling. This limitation is mitigated by the delay insertion method, leading to improvements of 10% and 9% shorter clock periods on average over conventional clock skew scheduling techniques for edge-sensitive and level-sensitive circuits, respectively. In mainstream digital circuit design flow, delay insertion is commonly used as a post-processing step in order to solve the short-path (hold time) violations. The drawbacks of delay insertion, such as increased circuit area and power consumption, are mainly disregarded in favor of the feasibility of the timing schedules. Similarly in this research, the drawbacks of delay insertion are considered tolerable in favor of the improvement in the circuit performance.

In Chapter 6, the LP automation framework of Chapter 4 is used to analyze an advanced multi-phase synchronization methodology with non-zero clock skew. This analysis is performed in order to provide design and analysis methods to address synchronous circuit design with emerging clocking technologies, some of which entail multi-phase synchronization schemes. For instance, the resonant rotary clocking technology provides an improved clock distribution network which satisfies the complex synchronization requirements of high-performance synchronous circuits by using multi-phase, non-zero clock skew clocking. The presented timing analysis method is the first to correctly capture the behavior of multi-phase, non-zero clock skew circuits in a fully-automated fashion. The experiments performed on ISCAS'89 benchmark circuits demonstrate that multi-phase synchronization can actually be advantageous in terms of circuit speed, despite the increased path delays due to latch insertion per each clock phase. Such a fact is contrary to common wisdom, which has over the years been suggested for zero clock skew systems. Approximately 17.7% and 12.0%

shorter clock periods are obtained on average over zero clock skew, edge-sensitive circuits for three-phase and four-phase synchronization schemes, respectively.

In Chapters 7 and 8, the integration of the presented timing and synchronization methodologies into the physical design flow of circuits synchronized with rotary clocking technology is described. Rotary clocking technology is a type of resonant clocking technology, which provides controllable skew, low-jitter, giga-hertz range clocking with fast transition times and low power consumption. Rotary clocking technology also permits non-zero clock skew operation and multi-phase synchronization of systems. In the presented research, the development of the physical design flow for rotary clock synchronized circuits is described. The physical design flow consists of a novel partitioning step in order to generate partitions of the circuit netlist on which clock skew scheduling can be applied individually. The potential to parallelize the application of clock skew scheduling is explored. Partitioning and the parallelization of the application of clock skew scheduling are shown to provide significant speedups in run times of the timing analysis. Over the ISCAS'89 benchmark circuits, a average speedup of 2.6x is observed. The speedup is shown to be directly proportional to the size of the circuit. Thus, when applicable, clock skew scheduling of partitions significantly improves the scalability of clock skew scheduling.

In summary, this work presents valuable timing and synchronization methodologies and their automation methods. The timing and synchronization methodologies are proposed especially for the non-zero clock skew operation of high-performance digital VLSI integrated circuits. The presented design automation algorithms are successfully integrated in the physical design flow of circuits synchronized with the rotary clocking technology. Conventional physical design flow steps are modified and alternative steps are described for the implementation of the proposed set of next-generation synchronous circuits. A CAD tool (hpictiming) is developed in order to implement this novel physical design flow.

10.0 FUTURE WORK

In this chapter, potential directions for future work are discussed. The discussion of advanced timing and synchronization of high performance integrated circuits presented in this dissertation are only a fraction of the current state-of-the-art. The potential directions for future research presented here are those, which are believed to have immediate and direct impact on the methodologies presented in this dissertation.

Two main directions of future research are presented. In Section 10.1, possible extensions to the hpictiming CAD tool are discussed. In Section 10.2, the adoptability of LP decomposition techniques to solving the clock skew scheduling problem of partitions is described.

10.1 EXTENSIONS TO THE CAD TOOL

As indicated in Chapter 8, the development of the hpictiming CAD tool is currently under progress. The road map for the development of the CAD tool suggests the integration of partitioning, clock skew scheduling and placement in one tool. The partitioning and clock skew scheduling portions of the tool are completed and discussed in detail in this dissertation. In order to complete the automation flow, the placement step must be completed.

The partitioning step of the hpictiming tool encompasses the placement of registers and logic network underneath and inside the ROA rings, respectively. This logic flow of the placement step is discussed in Section 8.1.5. In this flow, the registers are pre-placed in register banks underneath the ROA rings, designating multiple registers for each grain of clock delay. After partitioning and clock skew scheduling are performed, the registers in the netlist are *mapped* onto register banks. The mapping process is to be completed iteratively,

however, alternative methodical treatments are also possible. The size of register banks, in providing sufficient number of registers per phase and sufficient space for logic placement, is paramount to the success the mapping process.

The placement of logic network in the space inside the ROA rings also requires methodical treatment. The implementation of this phase is under development, and is a major milestone in the electronic design automation for circuits synchronized with the resonant rotary clocking technology.

Computation of the timing information of an integrated circuit is a feature that is currently not implemented in the hpictiming tool. In order to perform timing-driven partitioning or conventional clock skew scheduling, an accurate timing information of the circuit is required. As discussed in Chapter 8, the CAD tool is supplied with the initial timing information of a design through an SDF file generated by an external timing analysis program. For the analysis of ISCAS'89 benchmark circuits in experimentation (Section 8.3), for instance, the timing information is generated with an approximation algorithm and supplied to hpictiming. A very valuable extension to the hpictiming tool would be the capability to extract and compute the interconnect and gate delays of an integrated circuit. Such capability will not only enable hpictiming to be independent of third party tools, it will also enable hpictiming to be used as a stand alone static timing analysis tool.

Finally, the development of hpictiming can be unified. Currently, hpictiming is developed across multiple platforms, and is composed of multiple modules. The overall execution speed and accuracy (due to external timing information) of hpictiming depend partially on third party tools. The development of all hpictiming modules can be completed within the open source project, leading to a coherent, high-performance design and analysis tool. Such improvement will provide industry-strength capability to handle multi-million gate designs within shorter execution times.

In summary, potential directions for the timing-driven physical design research include:

- Developing the CAD module to perform the placement of register banks and synthesized logic,

- Enhancing the current CAD tool to handle interconnect and gate delay modeling and computation,
- Integrating the design tools in an open source development environment where the final tool can handle multi-million gate designs in reasonable amounts of run time.

10.2 LP DECOMPOSITION FOR CLOCK SKEW SCHEDULING

As discussed in Section 8.1.2, the partitioning of circuits in the presented physical design flow is performed to address a series of factors. Partitioning a circuit is primarily a design decision that enables the development of a dedicated design flow for circuits which are synchronized by the rotary clocking technology. Advantageously, partitioning enables the application of clock skew scheduling on smaller circuit areas and bolsters its parallelized application. The application of clock skew scheduling on circuit partitions (sequentially or in parallel) using the heuristic method presented in Section 8.1.1 is discussed.

In this section, the similarities of an LP decomposition technique to the presented heuristic method are exploited. As a more robust solution technique, the application of Dantzig-Wolfe LP decomposition technique [19, 92] to solve the clock slew scheduling problem of a circuit is proposed as a potential direction for future work.

Clock skew scheduling is performed on circuit partitions that are built up of local data paths (register-to-register timing paths). Remember from Section 8.1.2 that the LP problems for the partitions are called *partition LP* and the LP problem containing the constraints for inter-partition local data paths is called the *top block LP*.

Dantzig-Wolfe LP decomposition suggests the decomposition of a large-scale LP problem into:

- A master problem, which is composed of constraints that involve any variable,
- One or more subproblems, which are composed of constraints that involve a subset of variables.

The decomposition is used to solve the subproblems independently (sequentially or in parallel). The results of each subproblem are reflected on the master problem to find an optimal solution for the original LP. The propagation of results between the master and subproblems are performed repetitively until a solution is found. A complete discussion of Dantzig-Wolfe LP decomposition can be found in most linear programming text books, such as [19, 92].

It is immediately obvious that, the partition LP problems are analogous to subproblems and the top block LP problem is analogous to the master problem of Dantzig-Wolfe LP decomposition. This feature can be used to solve the partition LP problems in the presented physical design flow. This application will enable the guaranteed feasibility of the clock skew scheduling problem. Remember that in experimentation, 65% of the ISCAS '89 benchmark circuits were feasible after the first iteration of the presented heuristic method (Section 8.3.3). Alternative solution techniques are proposed in Section 8.1.4 in order to address the remaining 35% infeasible circuits. The first of the alternative procedures, the reiteration procedure, is conceptually similar to the Dantzig-Wolfe decomposition. It is projected that, the heuristic method presented in this dissertation will be faster compared to the Dantzig-Wolfe decomposition, however, the feasibility of solution is guaranteed with the latter, provided sufficient time is allotted for its execution.

The application of clock skew scheduling in parallel is discussed implicitly within the development of the physical design flow for rotary clock synchronized circuits. However, the partitioning and application of clock skew scheduling in parallel can be performed for circuits that are synchronized with traditional clocking technologies, as well. Dantzig-Wolfe LP decomposition can be further explored in this context in future research.

APPENDIX A

NONLINEAR PROBLEM FORMULATION

This appendix demonstrates the Nonlinear Programming (NLP) model problem formulation of the clock period minimization problem. The circuit network shown in Figure 16 is investigated for the clock period minimization problem and the NLP model problem formulation is demonstrated. The circuit network in Figure 16 is presented in Figure 61 for convenience.

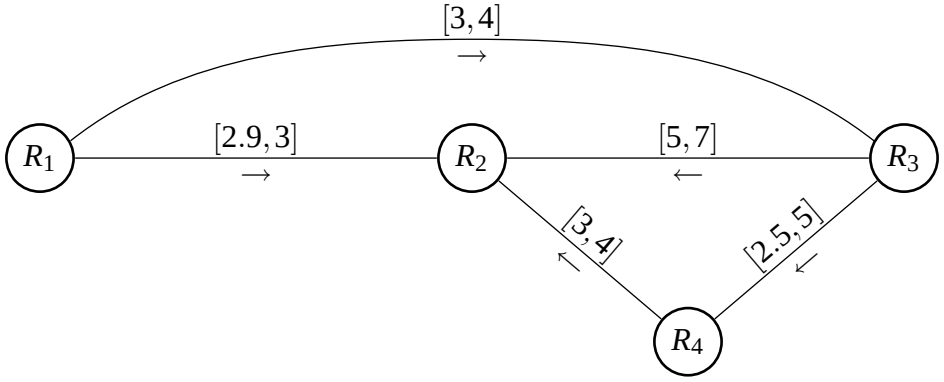


Figure 61: The simple synchronous circuit in Figure 16 (repeated).

(Obj) $Min \quad T$

such that

(i) Latching Constraints - Hold Time

$a_1 \geq 0$ $a_2 \geq 0$

$a_3 \geq 0$ $a_4 \geq 0$

(ii) Latching Constraints - Setup Time

$A_1 - T \leq 0$ $A_2 - T \leq 0$

$A_3 - T \leq 0$ $A_4 - T \leq 0$

(iii) Synchronization Constraints - Earliest Time

$d_1 = \max(a_1, 0.5T)$ $d_2 = \max(a_2, 0.5T)$

$d_3 = \max(a_3, 0.5T)$ $d_4 = \max(a_4, 0.5T)$

(iv) Synchronization Constraints - Latest Time

$D_1 = \max(A_1, 0.5T)$ $D_2 = \max(A_2, 0.5T)$

$D_3 = \max(A_3, 0.5T)$ $D_4 = \max(A_4, 0.5T)$

(v) Propagation Constraints - Earliest Time

$a_2 = \min\left[(d_1 + 2.9 + t_1 - t_2 - T), (d_3 + 5 + t_3 - t_2 - T), (d_4 + 3 + t_4 - t_2 - T)\right]$

$a_3 = d_1 + 3 + t_1 - t_3 - T$

$a_4 = d_3 + 2.5 + t_3 - t_4 - T$

(vi) Propagation Constraints - Latest Time

$A_2 = \max\left[(D_1 + 3 + t_1 - t_2 - T), (D_3 + 7 + t_3 - t_2 - T), (D_4 + 4 + t_4 - t_2 - T)\right]$

$A_3 = D_1 + 4 + t_1 - t_3 - T$

$A_4 = D_3 + 5 + t_3 - t_4 - T$

(vii) Validity Constraints - Arrival Time

$A_1 - a_1 \geq 0$ $A_2 - a_2 \geq 0$

$A_3 - a_3 \geq 0$ $A_4 - a_4 \geq 0$

(viii) Validity Constraints - Departure Time

$D_1 - d_1 \geq 0$ $D_2 - d_2 \geq 0$

$D_3 - d_3 \geq 0$ $D_4 - d_4 \geq 0$

(ix) Initialization Constraints

$A_1 = d_1$

APPENDIX B

LP PROBLEM FORMULATION

This appendix demonstrates the Linear Programming (LP) model problem formulation of the clock period minimization problem. The circuit network shown in Figure 16 is investigated for the clock period minimization problem and the LP model problem formulation[1] is derived. The circuit network in Figure 16 is presented in Figure 62 for convenience.

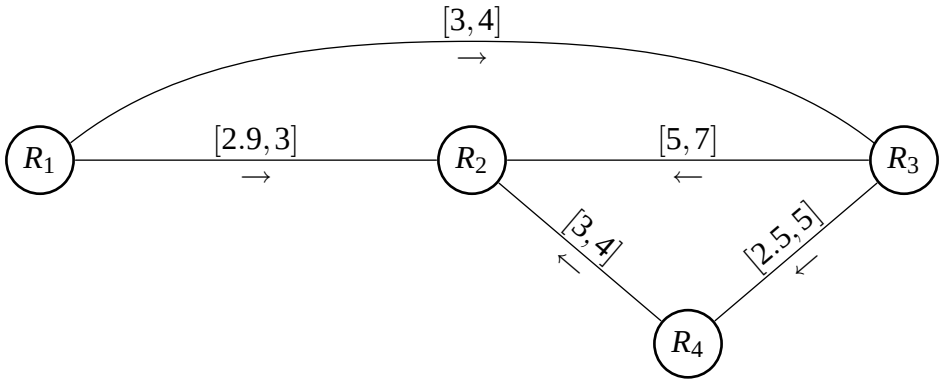


Figure 62: The simple synchronous circuit in Figure 16 (repeated).

(Obj) $Min \quad T + 1000d_1 + 1000d_2 + 1000d_3 + 1000d_4 + 1000D_1 + 1000D_2 + 1000D_3 + 1000D_4 + 1000A_2 + 1000A_3 + 1000A_4 - 1000a_2 - 1000a_3 - 1000a_4$

such that

[1]The constraints are labeled c1–c43 in order to improve the output readability.

(i) Latching Constraints - Hold Time

$c1 : a_1 \geq 0$ $\quad$ $c2 : a_2 \geq 0$

$c3 : a_3 \geq 0$ $\quad$ $c4 : a_4 \geq 0$

(ii) Latching Constraints - Setup Time

$c5 : A_1 - T \leq 0$ $\quad$ $c6 : A_2 - T \leq 0$

$c7 : A_3 - T \leq 0$ $\quad$ $c8 : A_4 - T \leq 0$

(iii) Synchronization Constraints - Earliest Time

$c9 : d_1 - a_1 \geq 0$ $\quad$ $c10 : d_1 - 0.5T \geq 0$

$c11 : d_2 - a_2 \geq 0$ $\quad$ $c12 : d_2 - 0.5T \geq 0$

$c13 : d_3 - a_3 \geq 0$ $\quad$ $c14 : d_3 - 0.5T \geq 0$

$c15 : d_4 - a_4 \geq 0$ $\quad$ $c16 : d_4 - 0.5T \geq 0$

(iv) Synchronization Constraints - Latest Time

$c17 : D_1 - A_1 \geq 0$ $\quad$ $c18 : D_1 - 0.5T \geq 0$

$c19 : D_2 - A_2 \geq 0$ $\quad$ $c20 : D_2 - 0.5T \geq 0$

$c21 : D_3 - A_3 \geq 0$ $\quad$ $c22 : D_3 - 0.5T \geq 0$

$c23 : D_4 - A_4 \geq 0$ $\quad$ $c24 : D_4 - 0.5T \geq 0$

(v) Propagation Constraints - Earliest Time

$c25 : a_2 - d_1 - t_1 + t_2 + T \leq 2.9$ $\quad$ $c26 : a_3 - d_1 - t_1 + t_3 + T \leq 3$

$c27 : a_2 - d_3 - t_3 + t_2 + T \leq 5$ $\quad$ $c28 : a_4 - d_3 - t_3 + t_4 + T \leq 2.5$

$c29 : a_2 - d_4 - t_4 + t_2 + T \leq 3$

(vi) Propagation Constraints - Latest Time

$c30 : A_2 - D_1 - t_1 + t_2 + T \geq 3$ $\quad$ $c31 : A_3 - D_1 - t_1 + t_3 + T \geq 4$

$c32 : A_2 - D_3 - t_3 + t_2 + T \geq 7$ $\quad$ $c33 : A_4 - D_3 - t_3 + t_4 + T \geq 5$

$c34 : A_2 - D_4 - t_4 + t_2 + T \geq 4$

(vii) Validity Constraints - Arrival Time

$c35 : A_1 - a_1 \geq 0$ $\quad$ $c36 : A_2 - a_2 \geq 0$

$c37 : A_3 - a_3 \geq 0$ $\quad$ $c38 : A_4 - a_4 \geq 0$

(viii) Validity Constraints - Departure Time

$c39 : D_1 - d_1 \geq 0$ $\quad$ $c40 : D_2 - d_2 \geq 0$

$c41 : D_3 - d_3 \geq 0$ $\quad$ $c42 : D_4 - d_4 \geq 0$

(ix) Initialization Constraints

$c43 : A_1 - d_1 = 0$

APPENDIX C

LP PROBLEM SOLUTION - CPLEX OUTPUT

This appendix includes the solution of the LP model problem describing the clock period minimization problem of the circuit network shown in Figure 62. The LP model problem shown in Appendix B is solved using the industrial solver CPLEX [36] and the results are shown below. In the results, `SECTION 1 - ROWS` section presents the optimal solution for each constraint and `SECTION 2 - COLUMNS` section presents the optimal results for each variable.

Note that the optimal objective function value is not completely relevant to the clock period minimization problem. Obtaining the minimum value for the clock signal period is the main objective of the clock period minimization problem and the minimum clock period is presented in `SECTION 2` ($T = 4.05$). Likewise, the optimal values for the data signal arrival and departure times are presented in `SECTION 2`, constituting the optimal clocking and timing schedules for the synchronous circuit under investigation. For detailed information about CPLEX operation and output formatting, see [36].

```
PROBLEM NAME     fig7.lp
 DATA    NAME
 OBJECTIVE VALUE  26254.05
 STATUS           OPTIMAL SOLN
 ITERATION        27

 OBJECTIVE        obj                 (MIN)
 RHS
 RANGES
 BOUNDS

 SECTION 1 - ROWS

  NUMBER .ROW...   AT  .ACTIVITY...  SLACK ACTIVITY  .LOWER LIMIT.  .UPPER LIMIT.  .DUAL ACTIVITY
```

1	obj	BS	26254.05	-26254.05	NONE	NONE	1
2	c43	EQ	0	0	0	0	-0
3	c1	BS	0	0	0	NONE	0
4	c5	BS	-2.025	2.025	NONE	0	-0
5	c9	BS	2.025	-2.025	0	NONE	0
6	c10	BS	0	-0	0	NONE	0
7	c17	BS	0	-0	0	NONE	0
8	c18	LL	0	0	0	NONE	-2000
9	c35	BS	2.025	-2.025	0	NONE	0
10	c39	LL	0	0	0	NONE	-2500.5
11	c26	UL	3	0	NONE	3	1000
12	c31	LL	4	0	4	NONE	-3500.5
13	c25	UL	2.9	0	NONE	2.9	2500.5
14	c32	BS	6.95	-3.95	3	NONE	0
15	c4	BS	0	0	0	NONE	0
16	c8	BS	-1.55	1.55	NONE	0	-0
17	c15	BS	2.025	-2.025	0	NONE	0
18	c16	LL	0	0	0	NONE	-1000
19	c23	LL	0	0	0	NONE	-1000
20	c24	BS	0.475	-0.475	0	NONE	0
21	c38	BS	2.5	-2.5	0	NONE	0
22	c23	BS	0.475	-0.475	0	NONE	0
23	c29	BS	2.475	0.525	NONE	3	-0
24	c34	BS	6.05	-2.05	4	NONE	0
25	c2	BS	0	0	0	NONE	0
26	c6	UL	0	0	NONE	0	500.5
27	c11	BS	2.025	-2.025	0	NONE	0
28	c12	LL	0	0	0	NONE	-1000
29	c19	LL	0	0	0	NONE	-1000
30	c20	BS	2.025	-2.025	0	NONE	0
31	c36	BS	4.05	-4.05	0	NONE	0
32	c40	BS	2.025	-2.025	0	NONE	0
33	c3	BS	1.025	-1.025	0	NONE	0
34	c7	BS	-2.025	2.025	NONE	0	-0
35	c13	BS	1	-1	0	NONE	0
36	c14	BS	0	-0	0	NONE	0
37	c21	LL	0	0	0	NONE	-2500.5
38	c22	LL	0	0	0	NONE	-2000
39	c37	BS	1	-1	0	NONE	0
40	c41	LL	0	0	0	NONE	-1000
41	c27	BS	2.95	2.05	NONE	5	-0
42	c28	UL	2.5	0	NONE	2.5	2000
43	c32	LL	7	0	7	NONE	-2500.5
44	c33	LL	5	0	5	NONE	-2000

SECTION 2 - COLUMNS

NUMBER	.COLUMN.	AT	.ACTIVITY...	..INPUT COST..	.LOWER LIMIT.	.UPPER LIMIT.	.REDUCED COST.
45	T	BS	4.05	1	0	NONE	0
46	D1	BS	2.025	1000	0	NONE	0
47	BD1	BS	2.025	1000	0	NONE	0
48	A4	LL	0	-1000	0	NONE	1000
49	BA4	BS	2.5	1000	0	NONE	0
50	D4	BS	2.025	1000	0	NONE	0
51	BD4	BS	2.5	1000	0	NONE	0
52	A2	LL	0	-1000	0	NONE	1500.5
53	BA2	BS	4.05	1000	0	NONE	0
54	D2	BS	2.025	1000	0	NONE	0
55	BD2	BS	4.05	1000	0	NONE	0
56	A3	BS	1.025	-1000	0	NONE	0
57	BA3	BS	2.025	1000	0	NONE	0
58	D3	BS	2.025	1000	0	NONE	0
59	BD3	BS	2.025	1000	0	NONE	0
60	BA1	BS	2.025	0	0	NONE	0
61	A1	LL	0	0	0	NONE	0
62	T1	BS	0.05	0	0	NONE	0

63	T3	LL	0	0	0	NONE	0
64	T2	BS	0.925	0	0	NONE	0
65	T4	BS	0.475	0	0	NONE	0

APPENDIX D

CHACO RUN SAMPLE

The following is a sample run output from *hpictiming*, specifically demonstrating the execution of the partitioning tool Chaco. In the run, the circuit netlist information is written to file `dabble.graph` file in the graph input file format required by Chaco. Chaco is provided by a set of user parameters from an input file `Users_Params`. Details about these parameters can be found in the Chaco User's Guide [31]. In the following excerpt, *hpictiming* is run on a very small sample circuit, with 11 circuit components.

```
Generate the graph file for partitioning? <y/n/h> [or <h> for help] : y

The selected option will write a graph file in the CHACO format for this circuit

*********************************************************
  The chaco graph file is written to file "dabble.graph".
*********************************************************

Run Chaco to complete partitioning <y/n> : y

===            CHACO OUTPUT START            ===


                    Chaco 2.0
          Sandia National Laboratories

Reading parameter modification file 'User_Params'
  Parameter 'OUTPUT_ASSIGN' reset to True
  Parameter 'ARCHITECTURE' reset to 2
  Parameter 'TERM_PROP' reset to True
  Parameter 'CUT_TO_HOP_COST' reset to 0.1
  Parameter 'COARSE_NLEVEL_KL' reset to 1
  Parameter 'KL_BAD_MOVES' reset to 2000
  Parameter 'KL_NTRIES_BAD' reset to 5
  Parameter 'KL_IMBALANCE' reset to 0.1
  Parameter 'MATCH_TYPE' reset to 3
  Parameter 'COARSEN_VWGTS' reset to True
  Parameter 'HEAVY_MATCH' reset to True
  Parameter 'REFINE_PARTITION' reset to 2
  Parameter 'INTERNAL_VERTICES' reset to 1
```

```
Graph input file: Assignment output file: Global partitioning method:
  (1) Multilevel-KL
  (2) Spectral
  (3) Inertial
  (4) Linear
  (5) Random
  (6) Scattered
  (7) Read-from-file
Number of vertices to coarsen down to: X and Y extent of of 2-D mesh: Partitioning dimension:
  (1) Bisection
  (2) Quadrisection


           Input and Parameter Values

Graph file: 'dabble.graph', # vertices = 11, # edges = 13
Global method: Multilevel-KL
Number of vertices to coarsen down to: 4
Eigen tolerance: 0.001
Local method: Kernighan-Lin
Partitioning target: 2-dimensional mesh of size 2x2
Partitioning mode: Bisection
Random seed: 7654321
Assignment output file: 'dabble.out' (normal format)
Active Parameters:
  CHECK_INPUT = True
  LANCZOS_TYPE:  Selective orthogonalization OR extended
  EIGEN_TOLERANCE = 0.001
  SRESTOL = -1 ... autoset to square of eigen tolerance
  LANCZOS_MAXITNS = -1 ... autoset to twice # vertices
  LANCZOS_SO_PRECISION = 2 ... double precision
  LANCZOS_SO_INTERVAL = 10
  LANCZOS_CONVERGENCE_MODE = 0 ... residual tolerance
  BISECTION_SAFETY = 10
  LANCZOS_TIME = 0 ... no detailed timing
  WARNING_EVECS = 2
  MAPPING_TYPE = 1 ... min-cost assignment
  MAKE_CONNECTED = True
  PERTURB = False
  COARSEN_RATIO_MIN = 0.7
  COARSE_NLEVEL_KL = 1
  MATCH_TYPE = 3
  HEAVY_MATCH = True
  COARSE_KL_BOTTOM = True
  COARSEN_VWGTS = True
  COARSEN_EWGTS = True
  KL_ONLY_BNDY = True
  KL_RANDOM = True
  KL_METRIC = Hops
  KL_NTRIES_BAD = 5
  KL_BAD_MOVES = 2000
  KL_UNDO_LIST = True
  KL_IMBALANCE = 0.1
  TERM_PROP = True
    CUT_TO_HOP_COST = 0.1
  OUTPUT_METRICS = 2
  MAKE_VWGTS = False
  REFINE_MAP = False
  REFINE_PARTITION = 2
  INTERNAL_VERTICES = True
  DEBUG_PARAMS = 2



Starting to partition ...

WARNING: Coarsening not making enough progress, nvtxs = 7, cnvtxs = 5.
         Recursive coarsening being stopped prematurely.
```

```
                    Partitioning Results

After full partitioning  (nsets = 4)

                           Total      Max/Set      Min/Set
                           -----      -------      -------
Set Size:                     11            3            2
Edge Cuts:                   402          202          200
Mesh Hops:                   403          203          200
Boundary Vertices:            12            4            2
Boundary Vertex Hops:         14            5            2
Adjacent Sets:                10            3            2
Internal Vertices:             4            1            1


Total time: 0.04 sec.
  partitioning 0.01
  other 0.03

KL time: 0.03 sec.
  nway refinement 0.03
    bucket sorting 0.01

Run Another Problem?


===            CHACO OUTPUT END            ===


Chaco executed, now reading back the results file.

Reading the chaco output can be done by two methods :
1) Speed  efficient
2) Memory efficient
Which method do you prefer? <1/2> : 1
Results read back succesfully.
```

BIBLIOGRAPHY

[1] C. Ababei, S. Navaratnasothie, K. Bazargan, and G. Karypis. Multi-objective circuit partitioning for cutsize and path-based delay minimization. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 181–185, 2002.

[2] C. Albrecht, B. Korte, J. Schietke, and J. Vygen. Cycle time and slack optimization for vlsi-chips. In *Digest of Technical Papers, IEEE/ACM International Conference on Computer-Aided Design*, pages 232–238, November 1999.

[3] G. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483 –485, 1967.

[4] W. Andress and D. Ham. Standing wave oscillators utilizing wave-adaptive tapered transmission lines. In *Digest of Technical Papers, 2004 Symposium on VLSI Circuits*, pages 50–53, June 2004.

[5] Apple Inc., Advanced Computing Group. *Xgrid Guide*, 2004.

[6] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley Publishing Company, Reading, MA, 1990.

[7] T. M. Burks and K. Sakallah. Optimization of critical paths in circuits with level-sensitive latches. In *Proceedings of ACM International Conference on Computer-Aided Design*, pages 468–473, 1994.

[8] T. M. Burks, K. A. Sakallah, and T. N. Mudge. Critical paths in circuits with level-sensitive latches. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 3(2):273–291, June 1995.

[9] Cadence Inc. *SOC Encounter v4.1 Manual*, May 2004.

[10] S. C. Chan, P. J. Restle, N. K. James, and R. L. Franch. A 4.6 ghz resonant global clock distribution network. In *IEEE ISSCC Digest of Technical Papers*, pages 341–343, February 2004.

[11] S. C. Chan, K. L. Shepard, and P. J. Restle. Design of resonant global clock distributions. In *Proceedings of the International Conference on Computer Design*, pages 238–243, 2003.

[12] W. K. Chen, editor. *The VLSI Handbook*. CRC Press, 1999.

[13] V. L. Chi. Salphasic distribution of clock signals for synchronous systems. *IEEE Transactions on Computers*, 43(5):597–602, May 1994.

[14] H. G. Chyun and J. Hung. Phase-locked loop techniques. a survey. *IEEE Transactions on Industrial Electronics*, 43(6):609–615, December 1996.

[15] M. R. Dagenais and N. C. Rumin. On the calculation of optimal clocking parameters in synchronous circuits with level-sensitive latches. *IEEE Transactions on Computer-Aided Design*, CAD-8(3):268–278, March 1989.

[16] J. Denker. A review of adiabatic computing. In *Proceedings of the 1994 Symposium on Low Power Electronics*, pages 94–97, October 1994.

[17] A. Drake, K. Nowka, T. Nguyen, J. Burns, and R. Brown. Resonant clocking using distributed parasitic capacitance. *IEEE Journal of Solid-State Circuits*, 39(9):1520–1528, September 2004.

[18] C. Ebeling and B. Lockyear. On the performance of level-clocked circuits. In *Proceedings of the Sixteenth Conference on Advanced Research in VLSI*, pages 342–356, March 1995.

[19] S.-C. Fang and S. Puthenpura. *Linear Optimization and Extensions: Theory and Algorithms*. AT&T. Prentice Hall, 1993.

[20] C. M. Fiduccia and R. Mattheyses. A linear heuristic for improving network partitions. In *Proceedings of 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[21] J. P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, C–39(7):945–951, July 1990.

[22] B. Floyd, X. Guo, J. Caserta, T. Dickson, C.-M. Hung, K. Kim, and K. O. Wireless interconnects for clock distribution. In *Proceedings of the 8th ACM/IEEE Intl. Workshop on Timing Issues in the Specification and Synthesis of Digital Systems,*, December 2002.

[23] B. Floyd, C. Hung, and K.K.O. Intra-chip wireless interconnect for clock distribution implemented with integrated antennas, receivers, and transmitters. *IEEE Journal of Solid-State Circuits*, 37(5):522–543, May 2002.

[24] W. Ford and W. Topp. *Data Structures with C++*. Prentice Hall, 1996.

[25] Free Software Foundation (FSF), http://www.gnu.org/software/glpk/glpk.html. *GLPK (GNU Linear Programming Kit)*, 2005. version 4.8.

[26] E. G. Friedman. *Clock Distribution Networks in VLSI Circuits and Systems*. IEEE Press, 1995.

[27] P. Gronowski and W. Bowhill. Dynamic logic and latches ii. *IEEE VLSI Circuits Workshop*, 1996.

[28] L. Hall, M.Clemens, W. Liu, and G. Bilbro. Clock distribution using cooperative ring oscillators. In *Proceedings of the 1997 Conference on Advanced Research in VLSI*, pages 15–16, September 1997.

[29] D. Harris and M. Horowitz. Skew-tolerant domino circuits. *IEEE Journal of Solid-State Circuits*, 32(11):1702–1711, November 1997.

[30] S. Held, B. Korte, J. Massberg, M. Ringe, and J. Vygen. Clock scheduling and clocktree construction for high performance asics. In *Proceedings of the International Conference on Computer Aided Design*, pages 232–239, November 2003.

[31] B. Hendrickson and R. Leland. The chaco user's guide: Version 2.0. Technical report, Sandia National Laboratories, Albuquerque, NM, Jul 1995.

[32] B. Hendrickson and R. W. Leland. A multi-level algorithm for partitioning graphs. In *Supercomputing*, 1995.

[33] P. Hofstee, N. Aoki, D. Boerstler, P. Coulman, S. Dhong, B. Flachs, N. Kojima, O. Kwon, K. Lee, D. Meltzer, K. Nowka, J. Park, J. Peter, S. Posluszny, M. Shapiro, J. Silberman, O. Takahashi, and B. Weinberger. A 1 ghz single-issue 64 b powerpc processor. In *Digest of Technical Papers of Solid-State Circuits Conference (ISSCC)*, pages 92–93, February 2000.

[34] Y. C. Hsu, S. Sun, D. Du, and X. Chu. Enhancing circuit performance under a multiple-phase clocking scheme. In *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, pages 219–222, June 1998.

[35] http://public.itrs.net/. International technology roadmap for semiconductors. Technical report, ITRS, 2002.

[36] ILOG, France. *CPLEX 7.1 User's Manual*, 2001.

[37] K. K. Nose and M. Mizuno. Parallel clocking: a multi-phase clock-network for 10ghz soc. In *Digest of Technical Papers. ISSCC. 2004 IEEE International Solid-State Circuits Conference*, volume 1, pages 344–531, February 2004.

[38] S. M. Kang and Y. Leblebici. *CMOS Digital Integrated Circuits: Analysis and Design*. The McGraw-Hill Companies, Inc., 1996.

[39] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 29:291–307, 1970.

[40] K. S. Kim and M. Papaefthymiou. Single-phase source-coupled adiabatic logic. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 97–99, 1999.

[41] J. T. Kong. Cad for nanometer silicon design challenges and success. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(11):1132–1147, November 2004.

[42] I. S. Kourtev and E. G. Friedman. A quadratic programming approach to clock skew scheduling for reduced sensitivity to process parameter variations. In *Proceedings of the 1999 IEEE ASIC/SOC Conference*, 1999.

[43] I. S. Kourtev and E. G. Friedman. *Timing Optimization Through Clock Skew Scheduling*. Kluwer Academic Publishers, 2000.

[44] N. Kurd, J. Barkarullah, R. Dizon, T. Fletcher, and P. Madland. A multigigahertz clocking scheme for the pentium(r) 4 microprocessor. *IEEE Journal of Solid-State Circuits*, 36(11):1647–1653, November 2001.

[45] J. Lee, D. T. Tang, and C. K. Wong. A timing analysis algorithm for circuits with level-sensitive latches. *IEEE Transactions on Computer-Aided Design*, CAD-15(5):535–543, May 1996.

[46] C. Leiserson and J. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1), 1991.

[47] R. Li, X. Guo, and K. O. A technique for incorporation of a heatsink for a system utilizing integrated circuits with wireless connections to an off-chip antenna. In *Proceedings of the IEEE 2004 International Interconnect Technology Conference*, pages 160–162, June 2004.

[48] I. Lin, J. A. Ludwig, and K. Eng. Analyzing cycle stealing on synchronous circuits with level-sensitive latches. *Proceedings of the 29th ACM/IEEE Design Automation Conference*, pages 393–398, June 1992.

[49] B. Lockyear and C. Ebeling. Optimal retiming of level-clocked circuits using symmetric clock schedules. *IEEE Transactions on Computer-Aided Design*, CAD-13(9):1097–1109, Sep 1994.

[50] I. Lustig. Private communication, 2004. ILOG Inc.

[51] N. Maheshwari and S. Sapatnekar. A practical algorithm for retiming level-clocked circuits. In *Proceedings of International Conference on VLSI in Computers and Processors*, pages 440–445, October 1996.

[52] MPI Standard Forum, http://www-unix.mcs.anl.gov/mpi/standard.html. *Message Passing Interface Standard v 2.0*, 1997.

[53] S. Naffziger, G. Colon-Bonet, T. Fischer, R. Riedlinger, T. Sullivan, and T. Grutkowski. The implementation of the itanium 2 microprocessor. *IEEE Journal of Solid-State Circuits*, 37(11):1448–1460, November 2002.

[54] F. O'Mahony, C. Yue, M. Horowitz, and S. Wong. A 10-ghz global clock distribution using coupled standing-wave oscillators. *IEEE Journal of Solid-State Circuits*, 38(11):1813–1820, November 2003.

[55] F. O'Mahony, C. P. Yue, M. Horowitz, and S. Wong. Design of a 10ghz clock distribution network using coupled standing wave oscillators. In *Proceesings of IEEE/ACM International Design Automation Conference*, pages 682–687, Anaheim, CA, June 2003.

[56] M. C. Papaefthymiou and K. Randall. Edge-triggering vs. two-phase level-clocking. In *Proceedings of the 1993 in Research in Integrated Systems*, March 1993.

[57] A. Pothen, H. Simon, and K. Liou. Partitioning sparse matrices eigenvectors of graphs. *SIAM Journal of Matrix Analysis*, 11:430–452, 1990.

[58] D. A. Pucknell and K. Eshraghian. *Basic VLSI Design*. Prentice Hall, 1994.

[59] J. M. Rabaey, A. Chadrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, Inc., Upper Saddle River, NJ, second edition, 2003.

[60] K. Ravindran, A. Kuehlmann, and E. Sentovich. Multi-domain clock skew scheduling. In *Proceedings of the International Conference on Computer Aided Design*, pages 801–808, November 2003.

[61] P. Restle. Resonant clock networks. http://www.research.ibm.com/, 2005. IBM Research, Computer Science, Innovative Matters, VLSI Design.

[62] P. J. Restle, T. G. McNamara, P. J. Camporese, K. F. Eng, K. A. Jenkins, D. H. Allen, M. J. Rohn, M. P. Quaranta, D. W. Boerstler, C. J. Alpert, C. A. Carter, R. N. Bailey, J. G. Petrovik, B. L. Krauter, , and B. D. McCredie. A clock distribution network for microprocessors. *IEEE Journal of Solid-State Circuits*, 36:792–799, May 2001.

[63] M. Saint-Laurent, M. Swaminathoan, and J. Meindl. On the micro-architectural impact of clock distribution using multiple plls. In *Proceedings of IEEE International Conference on Computer Design*, pages 214–220, September 2001.

[64] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. $checkT_c$ and $minT_c$: Timing verification and optimal clocking of synchronous digital circuits. *Proceedings of the IEEE/ACM International Conference on Computer–Aided Design*, pages 552–555, November 1990.

[65] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. Analysis and design of latch-controlled synchronous digital circuits. *IEEE Transactions on Computer-Aided Design*, CAD-11(3):322–333, March 1992.

[66] A. S. Sedra and K. C. Smith. *Microelectronic Circuits*. Oxford University Press, 1998.

[67] N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Graph algorithms for clock schedule optimization. *Proceedings of the IEEE/ACM International Conference on Computer–Aided Design*, pages 132–136, November 1992.

[68] N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Minimum padding to satisfy short path constaints. In *Proceedings of the IEEE/ACM International Conference on Computer–Aided Design*, pages 156–161, November 1993.

[69] N. Shenoy and R. Rudell. Efficient implementation of retiming. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 226–233, 1994.

[70] B. Stroustrup. *The C++ Programming Language*. Addison Wesley, 2000.

[71] Synopsys Inc. *Primetime Manual*, 2002.

[72] Synopsys Inc. *Synopsys Online Documentation*, 2002.

[73] T. G. Syzmanski and N. Shenoy. Verifying clock schedules. *Proceedings of the IEEE/ACM International Conference on Computer–Aided Design*, pages 124–131, November 1992.

[74] T. G. Szymanski. Computing optimal clock schedules. *Proceedings of the 29th ACM/IEEE Design Automation Conference*, pages 399–404, June 1992.

[75] B. Taskin. Linearization of the timing analysis and optimization of level-sensitive synchronous circuits. Master's thesis, University of Pittsburgh, Pittsburgh, PA, May 2003.

[76] B. Taskin. High performance integrated circuit (hpic) timing software package v1.9. http://sourceforge.net/projects/hpictiming/, 2004.

[77] B. Taskin and I. S. Kourtev. Linear timing analysis of soc synchronous circuits with level-sensitive latches. In *Proceedings of the 2002 IEEE ASIC/SOC Conference*, pages 358–362, September 2002.

[78] B. Taskin and I. S. Kourtev. Performance optimization of single-phase level-sensitive circuits using time borrowing and clock skew scheduling. In *ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 111–118, 2002.

[79] B. Taskin and I. S. Kourtev. Advanced timing of level-sensitive sequential circuits. In *11th Annual IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Tel-Aviv, Israel, December 2004.

[80] B. Taskin and I. S. Kourtev. Delay insertion method in clock skew scheduling. *IEEE Transactions in Computer-Aided Design*, 2004. (in submission).

[81] B. Taskin and I. S. Kourtev. Linearization of the timing analysis and optimization of level-sensitive digital synchronous circuits. *IEEE Transantions on Very Large Scale Integration (VLSI) Systems*, 12(1):12–27, January 2004.

[82] B. Taskin and I. S. Kourtev. Multi-phase synchronization of level-sensitive circuits - a cad perspective. *IEEE Transactions on Computer-Aided Design*, 2004. (in revision).

[83] B. Taskin and I. S. Kourtev. Performance improvement of edge-triggered sequential circuits. In *11th Annual IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Tel-Aviv, Israel, December 2004.

[84] B. Taskin and I. S. Kourtev. Time borrowing and clock skew scheduling effects on multiphase level-sensitive circuits. In *Proceedings of the 2004 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume II, pages 617–620, Vancouver, Canada, May 2004.

[85] B. Taskin and I. S. Kourtev. Delay insertion in clock skew scheduling. In *ACM International Symposium on Physical Design (ISPD)*, San Francisco, CA, April 2005.

[86] B. Taskin, J. Wood, and I. S. Kourtev. Timing-driven physical design for digital synchronous vlsi circuits using resonant clocking. In *ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 53–59, San Francisco, CA, February–March 2005.

[87] J. P. Uyemura. *Introduction to VLSI Circuits and Systems*. John Wiley & Sons, Inc., 2002.

[88] J. Warnock. Circuit design issues for the power4 chip. In *Proceedings of the 2003 International Symposium on VLSI Technology, Systems, and Applications*, pages 125–128, October 2003.

[89] C. Webb, C. Anderson, L. Sigal, K. Shepard, J. Liptay, J.D.Warnock, B. Curran, B. Krumm, M. Mayo, P. Camporese, E. Schwarz, M. Farrell, P. Restle, R. A. III, T. Slegel, W. Houtt, Y. Chan, B. Wile, T. Nguyen, P. Emma, D. Beece, C. Ching-Te, and C. Price. A 400-mhz s/390 microprocessor. *IEEE Journal of Solid-State Circuits*, 32(11):1665–1675, November 1997.

[90] N. H. E. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, Reading, MA, third edition, 2004.

[91] R. Williams. Performance of dynamic lod balancing algorithms for unstructured mesh calculations. *Concurrency*, 3:457–481, 1991.

[92] W. L. Winston. *Operations Research Application and Algorithms*. PWS-Kent Publishing Company, second edition, 1991.

[93] J. Wood. Electronic circuitry. United States Patent Number 6,556,089, April 2003.

[94] J. Wood. Electronic circuitry. United States Patent Number 6,816,020, November 2004.

[95] J. Wood. Private communication, 2005. MultiGiG Inc.

[96] J. Wood, T. Edwards, and S. Lipa. Rotary traveling-wave oscillator arrays: a new clock technology. *IEEE Journal of Solid-State Circuits*, 36(11):1654–1665, November 2001.

[97] J. Wood, S. Lipa, P. Franzon, and M. Steer. Multi-gigahertz low-power low-skew rotary clock scheme. In *IEEE International Solid-State Circuits Conference, 2001. Digest of Technical Papers. ISSCC. 2001*, pages 400–401, February 2001.

[98] T. Xanthopoulos, D. Bailey, A. Gangwar, M. Gowan, A. Jain, and B. Prewitt. The design and analysis of the clock distribution network for a 1.2 ghz alpha microprocessor. In *Digest of Technical Papers. ISSCC. 2001 IEEE International Solid-State Circuits Conference*, pages 402–403, February 2001.

[99] H. Zhou. Clock schedule verification crosstalk. In *ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 78–83, 2002.